(2)

N PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 17, 1992	3. REPORT TYPE AND DATES COVERED (b) Final Technical Report

4. TITLE AND SUBTITLE

Intelligent Signal Processing for Active Control

5. FUNDING NUMBERS

C-N0001489-J-1633

6. AUTHOR(S)

P.A. Ramamoorthy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Cincinnati
Dept. of Electrical & Computer Eng.
Cincinnati, OH 45221-0030

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Program Manager: Dr. Eric Hendricks, Code 122
800 N. Quincy St.
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research is concerned with the use of neural architectures and fuzzy expert systems in nonlinear system identification and in the control of such systems. In particular, on-line identification/modeling is considered. The research has resulted in a technique where the network can evolve (in size) in time so as to provide an optimal model/controller. Also an adaptive algorithm, which is less sensitive to initial values of the weights and the learning rate, has been developed. We have also established a common framework between neural networks and fuzzy expert systems and developed a neuro-fuzzy architecture that retains the best of the two areas. The use of the architectures and the adaptation algorithm has been demonstrated on a number of applications.

14. SUBJECT TERMS

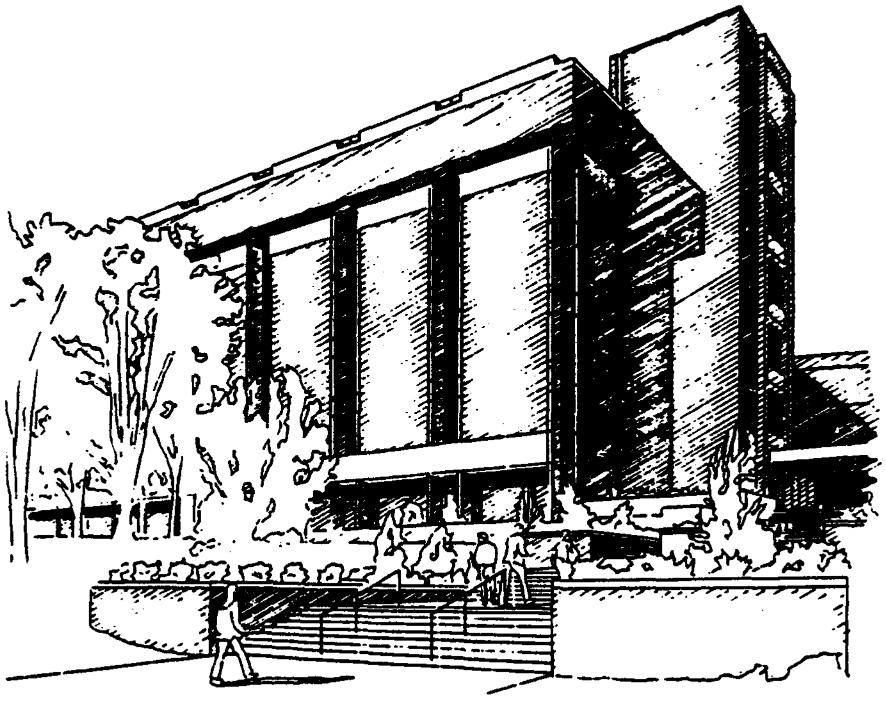
Neural Nets, Active Control, Fuzzy Logic, recurrent nets.

15. NUMBER OF PAGES

39

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT UL



UNIVERSITY OF CINCINNATI
COLLEGE OF ENGINEERING

92-16790

Intelligent Signal Processing For Active Control

P.A. Ramamoorthy
Department of Electrical & Computer Engineering

Final Report, Contract No. N0001489-J-1633

OFFICE OF NAVAL RESEARCH

Contract No: N0001489-J-1633 P.I.: P.A. Ramamoorthy

Intelligent Signal Processing For Active Control

1 Executive Summary

The thrust of this research is in the evaluation of neural architectures for nonlinear system identification and control of such systems. Also, it involved investigation into merging of neural architectures and fuzzy expert systems and the application of the hybrid architecture/approach in nonlinear system identification and control.

System identification is the process of selecting suitable models and identifying the parameters of the model to represent a system under consideration. System identification is necessary before a suitable controller can be designed to optimize the system performance. The model is selected through some apriori knowledge (if available) of the system or arbitrarily if no information is available. The parameters are estimated from a large number of input/output samples obtained from experiments on the system. The adaptation or training procedure optimizes the parameter values to minimize a chosen error or cost function. The adaptation may be off-line where all the I/O samples are assumed to be available simultaneously or on-line where one I/O sample is available at a time and the model parameters are adapted as and when they become available.

Linear models are well established for system identification since they are mathematically tractable. AR (autoregressive or all-pole) and ARMA (autoregressive moving average or pole-zero) models belong to this category. Even under this simple case, we assume that the order of the model is known and find the parameters from the training data. Volterra-Wiener methods, Block-oriented methods (Uryson, Hammerstein models) and self-organizing techniques (such as Group method of data handling) are three techniques commonly used for nonlinear system identification and each has certain disadvantages. For example, the Volterra-Wiener series can be thought of as a Taylor series with memory containing various cross-product terms of the input data to produce the nonlinear effect. The problem here is that proper nonlinear terms have to be included in the model before the parameters are estimated. Further, the number of parameters in the model increases exponentially as the nonlinear terms (say from 2-nd order to 3-rd order) are increased. Also, being a Taylor series, it cannot be applied to all practical systems (systems containing saturating nonlinearity, for example). The other two methods have similar problems.

Multi-layer neural networks can be thought of as models for arbitrary nonlinear mapping (f: x $\Rightarrow$ y, where x represents the input vector, y the output vector and f, the nonlinear transformation). The nonlinear mapping property is achieved by using tanh or sigmoidal nonlinearities as the nonlinear building blocks along with linear building blocks such as summation, weighting and bias. Since a tanh function corresponds to a ratio of two infinite degree polynomial in its argument, through addition of many such nonlinearities, layers and weights/summers/nodes, we create a highly nonlinear mapping without requiring any apriori information about the nonlinear terms that need to be present in the model. The weights provide the "degrees of freedom" necessary to obtain the desired mapping and by systematically increasing their number, we can increase the accuracy of mapping or modeling. Thus, it can be argued that neural networks can outperform other techniques in nonlinear system modeling.

There are some problems in using neural networks for system identification. The present approaches, assume a fixed structure (number of layers and number of nodes in each layer) and use the back propagation or derivatives of that algorithm to obtain the values of the weights. If such a structure doesn't work out, then one has to change the number of nodes etc (a trial and error procedure). Also, the back propagation algorithm is extremely slow, may not converge (whence the learning rate has to be adjusted and the adaptation procedure continued) and sensitive to the initial values of the weights. Such problems limit the use of neural networks in system identification and prohibit their use in on-line system identification.

Considering these problems, we have developed a methodology that will let the network evolve (in size) in time through the use of simple neural nets as building blocks. Such an approach makes the adaptation much faster and further eliminates the need to guess the correct number of nodes apriori and so on. It also makes the adaptation less sensitive to initial values of the weights and the learning rate. We have also modified a well known algorithm in linear system modeling known as the recursive least squares algorithms that has got faster converging properties (This algorithm also has high computational complexity which becomes an issue when a large number of weights are involved. The time evolving architecture helps to overcome that problem). Software for system modeling based on this approach has been developed and tested on data generated from different system models (linear, bilinear, saturating type nonlinearities and piece-wise linear). The results are indeed encouraging and demonstrated clearly that our new time evolving architecture and the algorithm could be used to model nonlinear systems without requiring any apriori information about the system structure, type of nonlinear etc. Of course, all our simulations assumed that there is some knowledge about the memory of the system and we need to make further simulations to see if that requirement can also be eliminated or at least its effects can be minimized[1].

We have also investigated the application of the new, time-evolving architecture and the adaptation algorithm in the design of controllers (model reference adaptive control). The results indicate that neural networks can effectively used to implement nonlinear controllers. The results of this investigation have been presented in a number of IEEE conferences (1-3 in list of publications). Copies of these papers are appended to this report as appendices.

Fuzzy expert systems (FES) based on fuzzy logic can also be considered as systems for functional mapping (linear or nonlinear). This observation led to a study of the pros and cons of the two approaches (NNs and FESs). We were able to show that fuzzy expert systems can be mapped into an architecture that incorporates a number of smaller and simpler multi-layer neural networks with structured interconnections between them. This implies that though theoretically a two-layer (two hidden layer) neural network can approximate any arbitrary mapping, in practice, a fuzzy expert system can provide better approximation. This can also be philosophically explained as follows: The neural network approach uses two attributes of a system, 1) Large input samples; 2) Corresponding output samples along with the training procedure to model the system. It doesn't use any other information even if such information is readily available. On the other hand, the FES approach, uses six attributes of a system, 1) Input variables, 2) Output variables, 3) Fuzzy sets of the input and output variables, 4) Membership functions corresponding to those fuzzy sets, 5) Fuzzy rule base which indicate the regions (fuzzy sets) to which the actual output(s) should belong given that the input(s) belong to certain fuzzy sets and 6) Defuzzification procedure to obtain the actual values of the output given the fact that they lie under certain output fuzzy sets and the membership functions. Thus, a FES uses much more information about the system than that of a NN and hence ought to perform much better. More importantly, information such as fuzzy sets and membership functions can be deduced from the problem under consideration[2] rather easily.

Based on such considerations, in our research, we were able to establish a common frame-work between NNs and FESs, and derive a hybrid approach that combines the general working philosophy behind FESs with the trainability "aspect of NN" to come up with a new "Neuro_Fuzzy Architecture" and methodology that retains the best of

[1] The time-evolving property of the new architecture can also be used to minimize the need for knowledge about the memory of the system.

[2] The choice that we make for fuzzy sets and membership functions may not be the same as the one used in the actual system, but they don't seem to make much difference and under such circumstances, it appears better to incorporate such information than incorporating none.

both worlds. We have completed initial experimentation with such an architecture and the methodology in the design an optimal controller for a specific problem (that of backing up a truck to the loading dock from any reasonable starting position). The results obtained are indeed encouraging. Copies of papers (4-6) describing the approach and the results are attached.

Another exciting fall-out from this research and which we are currently pursuing is that of the design of neural networks with feedback (also known as recurrent neural nets) and nonlinear system design. In this research, we showed that the neural networks (especially recurrent nets) can be approached from an entirely different angle[3] -- that of (or as analogues of) passive nonlinear networks. Such networks are formed by proper interconnections of various nonlinear elements where each and every nonlinear element is constrained to be lossless or lossy[4]. When energy storing elements (which themselves could be nonlinear) are present in such a network, we can obtain a (or set of) Input/Output relationship(s) as nonlinear differential equation(s). The basic property that the network is lossy (or consumes energy) ensures that the nonlinear differential equations obtained from the network would represent absolutely stable systems and this property will hold as long as the individual element values are maintained in their permissible range of values. Thus, to design complex nonlinear systems (a complex nonlinear plant plus a controller to optimize its performance, for example), one simply has to force the system dynamics to mimic the dynamics of a properly constructed passive nonlinear network, a process akin to reverse engineering.

In our investigation, which is in its early stages, we have utilized the above approach and applied it with relative ease to a number of problems leading to encouraging results. The fruits of such an approach seems to be endless. For example, the approach can be applied to linear and nonlinear controller design (for linear and nonlinear plants), self-tuning controllers, model reference adaptive controllers, Self-organizing networks, adaptive IIR filter design, adaptive beam forming, Two-dimensional systems, fuzzy systems etc. In this We made a presentation of this work recently (May 13, 92) at NOSC, San Diego.

---

[3]The neural nets concept has its origin in biological neural networks which have tremendous capabilities in terms of rapid/robust recognition, learning etc. It has been the expectation of the research community that by mimicking such networks one would be able to design complex systems with similar capabilities. However, a major problem in the design of recurrent neural networks is the question of stability and some ad hoc methods have been developed to overcome this problem.

[4] A number of new elements have been proposed for this purpose.

The two-year funding enabled the PI and the Co-I to partially support a number of graduate students. One former student, Dr. Phillip Pace now works for General Dynamics and is scheduled to join Naval Post-Graduate School in Sept. 1992 as an Assistant Prof. Govind Girish, Song Huang and Shi Zhang are expected to complete their Ph.D's before the end of this year.

2. List of Publications

1. G. Govind & P.A. Ramamoorthy, "A new time-evolving neural network architecture and algorithm for nonlinear system i identification using adaptive filtering techniques," *Proc. IEEE Intl. Symp. Circuits &Systems*, May 1992

2. G. Govind and P.A. Ramamoorthy, "Multi-layered neural networks for arbitrary approximation: An explanation and simulations,: *IEEE Intl. Conf. on Artificial Neural Networks in Engineering*, Nov. 1991.

3. P.A. Ramamoorthy & G. Govind, "Multi-layered neural networks and Volterra series: The missing link," *Proc. IEEE Intl. Conf. Systems Engineering*, Aug. 1990

4. P.A. Ramamoorthy, S. Zhang & S. Huang, "Adaptive fuzzy expert systems for control applications," *Proc. Intl. Fuzzy Systems & Intelligent Control Conf.*, Mar.1992.

5. P.A. Ramamoorthy & S. Huang, "Cerebellar model articulation controller neural network - A simple fuzzy expert system in disguise?," *IEEE Intl. Conf. on Artificial Neural Networks in Engineering*, Nov. 1991.

6. P.A. Ramamoorthy & S. Huang, "Fuzzy expert system Vs neural networks for truck-backer-upper control problem'" *IEEE Intl. Conf.Systems Engineering*, Aug. 1991.

3. Appendix (Copies of publications)

A New Time-evolving Neural Network Architecture and Algorithm for Nonlinear System Identification using Adaptive Filtering Techniques[1]

Girish Govind and P.A. Ramamoorthy
M. L. 30, Department of Electrical & Computer Engineering
University of Cincinnati, Cincinnati, OH 45221-0030

Abstract

Concepts from adaptive filtering and some heuristics are utilized to obtain a fast convergent online neural network especially suited for nonlinear system identification. Rather than training a fixed neural network structure, the algorithm presented allocates nodes when required. This provides for an optimal allocation of hidden nodes in this structure. The results obtained show that the neural network model presented is a viable approach for Nonlinear System Identification and can be applied to a large class of nonlinear systems. Simulations are provided that show the fast convergence of this neural network structure.

1. Introduction

The process of abstracting relationships between inputs and outputs using only the data observed from the system is called system identification[1]. Most of the recent developments have been in the area of linear system identification where a linear structure is assumed for the model. However, most physical systems are nonlinear and thus a nonlinear model is required to get a globally valid model (true for all inputs) for a system.

Artificial neural networks have received increasing attention in recent years. Neural models are composed of a highly interconnected mesh of nonlinear elements (neurons) whose structure is drawn from our current understanding of the biological neural system[2].

The main properties of multi-layered neural networks are their ability to learn from experience, generalize the performance over untrained inputs, abstract relationships between signals, and to arbitrarily approximate any map given sufficient number of neurons. Clearly these properties are the same as those of interest to researchers in the area of system identification. Although the above properties appear to be the solution to all the problems in conventional system identification, in actual practice there are other limitations that restrict their performance. For instance, convergence is not easy to obtain in multi-layered networks and may take several hundreds of thousands of iterations. Fine adjustment of the learning parameters and repeated presentations of the available data are required for satisfactory approximation. In online adaptation, the training is slow and may not converge. There is also no method (or heuristic) to select the number of hidden nodes for a good approximation – less numbers can lead to an imprecise model, excess numbers can make the training very slow and provide poor generalization after training[3].

This paper describes an algorithm that has been developed for performing online adaptation of the weights. Unlike the backpropagation algorithm that trains a fixed structure[2], in this algorithm the network is built slowly in a step-by-step fashion. This evolving architecture methodology permits a near optimal allocation of hidden nodes and at the same time, provides sufficient "degrees of freedom" for approximating a nonlinear system. Although the computational complexity of this algorithm per iteration could be higher than that of the standard backpropagation, it exhibits fast convergence. Also as the network is built during training, the complexity changes from iteration to iteration.

2. Nonlinear System Identification

System identification usually consists of two stages – *model selection*, and *parameter estimation*. In neural-based system identification, the selection of the number of hidden nodes corresponds to the model selection stage. The backpropagation algorithm utilizes gradient descent to determine the weights of the network and thus corresponds to the parameter estimation stage. Neural networks in system identification have not seen much change since they were first proposed by Lapedes and Farber[4] and also independently by us[5].

The next section presents the development of an adaptive neural algorithm that works quite differently from conventional designs. It has some similarities with another recent algorithm that performs model building combined with parameter estimation[6] but as will be shown with the chaotic prediction example, the presented algorithm is far more parsimonious in the number of parameters, and faster.

3. New Architecture and Algorithm

In a recent paper[3] it was discussed that the problem

[1]The work reported here has been supported by NASA Headquarters through the NASA-UC Space Engineering Center and Office of Naval Research Grant Number 00014-81-J-1633

of approximation in a multilayer neural network with binary neurons is NP-complete but if the network can allocate more nodes it may be possible to solve a problem in polynomial time, and thus we use an evolving architecture strategy. We restrict the scope of the presented algorithm to problems of nonlinear system identification where the input-output data is not binary.

A node is allocated when really required and the parameters are adjusted for other data points. The strategy of allocation and adaptation is also accompanied with a strategy for pruning as redundant or noncontributing nodes may be allocated while growing, or nodes may become so after adaptation. The next few paragraphs will provide some more detail about these new heuristics and in the next section these will be tested for some sample problems.

In this paper we use a node with a gaussian bar[7] activation function which is different from both the sigmoid and the radial basis functions as it has a semilocal response. The neural structure used for this algorithm is as shown in Figure 1. The bold lines are weighted connections while the dotted lines are only performing distribution of the signals. The larger circles represent nodes which perform a weighted summation of the inputs and then propagate the sum through an adjustable gaussian nonlinearity – these are referred to as hidden nodes, the small circles represent an adjustable gaussian nonlinearity – these will be called gaussian bar nodes. More of each of these get allocated during the training procedure. The output stage is linear and a parallel structure that has direct connections (i.e. no hidden nodes) to the output is also built. This total structure is used as a model for approximating the problem at hand. This can be written as

$$y_n = \sum_{m=1}^{M} \sum_{i=1}^{P(m)} w_{nmi} N_i(x_m) + \sum_{k=1}^{Q} w2_{nk} N2_k(v_k)$$

where

$$v_k = \sum_{m=1}^{M} \sum_{l=1}^{R(m)} w1_{ml} N1_l(x_m)$$

where the following notation has been used: x_m is one of the M inputs, y_n - one of the N outputs, w - the parallel direct structure weights, $w1$ – weights of the first layer, and $w2$ - weights of the second layer. The neural structure after training can be described by the three numbers, P(m) – number of gaussian bar nodes allocated for direct structure input m, Q – number of hidden nodes, and R(m) – number of gaussian bar nodes for first layer input m. N, $N1$, or $N2$ represent adjustable gaussian nonlinearities where both the center and the width is adjustable.

Two parameters are very important for the allocation process - a variable error threshold and a neighborhood parameter is used to decide when new hidden nodes, or new gaussian bar nodes should be allocated. These thresholds are set to a high value at the start of the simulation but are allowed to decrease slowly to a small value. The neighborhood parameter discourages the allocation of gaussian bar nodes very near to each other.

The complete detailed algorithm has not been flowcharted here as it would require a lot more space but the main ideas behind it are given here. If the modeling error at the output is greater than the variable error threshold value at the time, and the distance to the closest neighbor gaussian bar node is greater than the variable neighborhood threshold value then a new gaussian bar node is allocated. However, if the modeling error is much greater than the error threshold (several times over) then a new hidden node is allocated (for the two layer structure). If the error is less than the threshold then no new structure is allocated but all the weights, and the mean and variances of the hidden nodes and gaussian bar nodes are adjusted to move towards lowering the error.

The method to prune the gaussian bar nodes and the hidden nodes is to check if the weighting factors at their outputs are significant. This method is used to prune gaussian bar nodes in the first layer (and the parallel direct layer), and to prune hidden nodes in the second layer. Other conditions to check for are the mean of the gaussian bar node being adjusted and moved outside the range of the signal or the gaussian node becoming just a spike.

It is important to note that the above scheme is only a heuristic, like several other methods that are in use today to speed up the convergence. The heuristics presented in this paper combine the stages of model selection and parameter estimation and work better than other training algorithms. In The algorithm by Sanger[6] the network is grown at regular intervals and never pruned. Here the network is grown when desired, and pruned at regular intervals. This pruning procedure is very important as it improves generalization and also keeps the network size in check, thus yielding a small network that can solve the given problem.

4. Simulations

In this section, three examples are provided. Two examples are on online nonlinear system identification, and one on online chaotic series prediction. The presented performance is typical of the networks and the learning, and growing parameters are the same throughout the simulations. The software has not been optimized in any way at this time and yet each simulation takes no more than ten minutes on a Sun workstation.

System identification example 1

The following system is simulated and the network is allowed to grow using the algorithm described in the previous section.

$$y[k+1] = 0.8y[k] + (u[k] - 0.8)u[k](u[k] + 0.5)$$

Uniformly distributed random noise in the range [-1, 1] is used for the input $u[k]$. The modeling error is used to grow or adapt the network structure in an online fashion. As can be seen in figure 2 the network quickly learns the input-output mapping. The final network structure was $P(1) = 7$, $P(2) = 9$, $Q = 5$, $R(1) = 5$, and $R(2) = 3$. After this training, the parameters of the network were kept fixed and the network tested with the actual model data. This is shown in figure 3 and the actual and modeled output are not distinguishable.

System identification example 2

The system in this example is a multi-input, multi-output one with four inputs and two outputs.

$$\begin{bmatrix} y_1[k+1] \\ y_2[k+1] \end{bmatrix} = \begin{bmatrix} \frac{y_1[k]}{1+y_2^2[k]} \\ \frac{y_1[k]y_2[k]}{1+y_2^2[k]} \end{bmatrix} + \begin{bmatrix} u_1[k] \\ u_2[k] \end{bmatrix}$$

The two inputs $u_1[k]$ and $u_2[k]$ are random uniformly distributed in the range [-1, 1]. This model corresponds to a $M = 4$, $N = 2$ model, and after convergence the structure has the P values as 10, 12, 13, and 15. The number of hidden nodes was 25 (this was imposed on the model), and the R values were 21, 18, 13, and 15. After the model has been constructed the network output was compared against the actual model outputs. These are shown in figures 4 and 5. Here, the network outputs are not as exact as the other simulations but this is due to the restriction placed on the number of hidden nodes.

Chaotic prediction: Mackey-Glass ($\tau = 30$)

This is a classic benchmark problem for neural network training algorithms. This was first used by the Los Alamos nonlinear dynamics group and is now tested for by every known multi-layered training method.

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t)$$

A fourth-order Runge-Kutta is used to provide values of x at discrete time steps. The initial condition used is $x = 0.8$. This problem corresponds to a 6 input ($M = 6$), one output ($N = 1$) problem. The six inputs correspond to $x[t-6m]$, $m = 0..5$, and the desired output is $x[t+6]$. The training error squared is shown in figure 6 and exhibits the usual fast convergence. The actual series is plotted against the predicted series for a six-step ahead prediction in figure 7. Prediction further into the future can be made through iteration of the map. The network constructed here has a normalized performance index of 0.05 which is almost as exact as the simulation by Sanger[6]. In that paper there were 20 sinusoidal basis functions used for each of the 6 inputs and 106 trees were constructed i.e. the total number of estimated parameters was over 12,000 from about 42,000 samples. In contrast to that simulation, the algorithm presented in this paper constructed a network of P values as 5, 7, 5, 4, 4, and 7. There were 18 hidden nodes, and the R values were 8, 10, 9, 5, 6 and 10. Considering all the parameters this corresponds to less than 2000 parameters. Thus this algorithm is very parsimonious in the parameters.

5. Discussion and Conclusions

In this paper some heuristics have been presented that allow fast and online construction of nonlinear input-output maps from data. Unlike the usual multilayered neural network where apriori knowledge is difficult or impossible to incorporate into the network, in the presented algorithm it is straightforward.

It is expected that the presented algorithm will find use into some engineering applications which require on-line performance - particularly channel equalization, and nonlinear adaptive control.

References

[1] L. Ljung, *System Identification: Theory for the User.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

[2] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation.* Reading, MA: Addison-Wesley, 1991.

[3] E. B. Baum and D. Haussler, "What size net gives valid generalization," *Neural Computation*, vol. 1, pp. 151–160, 1989.

[4] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: Prediction and system modeling," tech. rep., Los Alamos National Laboratory, 1987.

[5] P. A. Ramamoorthy, G. Govind, and V. Iyer, "Signal modeling and prediction using neural networks," in *International Neural Network Society First Annual Meeting*, (Boston, MA), September 1988.

[6] T. D. Sanger, "A tree-structured adaptive network for function approximation in high-dimensional spaces," *IEEE Trans. on Neural Networks*, vol. 2, pp. 285–293, March 1991.

[7] E. Hartman and J. D. Keeler, "Predicting the future: Advantages of semilocal units," *Neural Computation*, vol. 3, pp. 566–578, 1991.

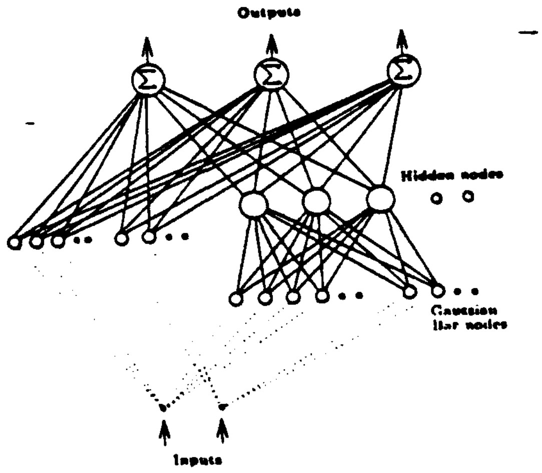


Figure 1. Neural Network structure.

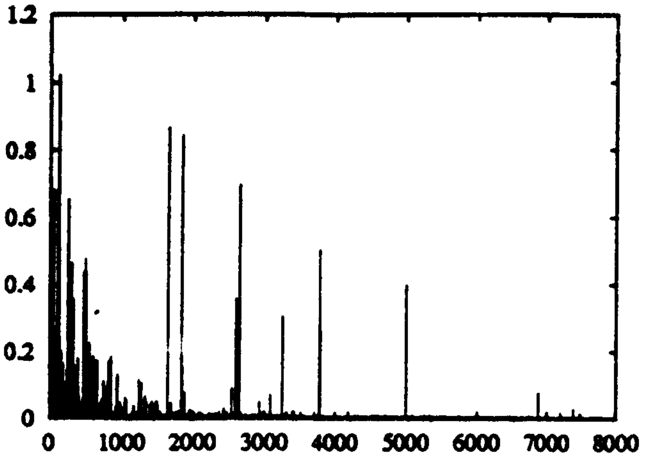


Figure 2. Identification example 1: Modeling error (squared) during training.

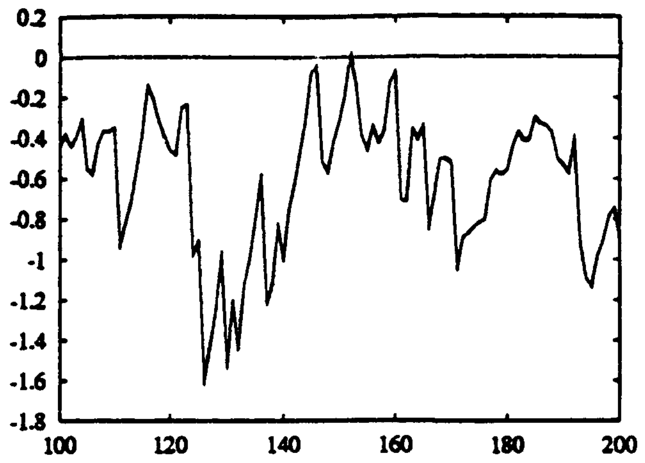


Figure 3. Identification example 1: Comparison between the actual model output and the network output.

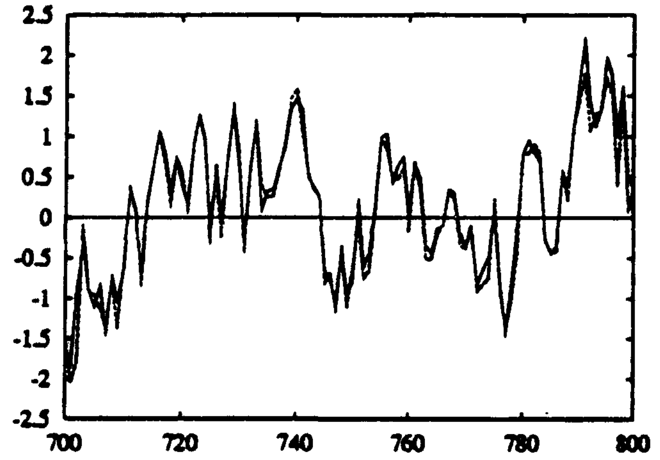


Figure 4. Identification example 2, Output # 1: Comparison between actual model output and network output.

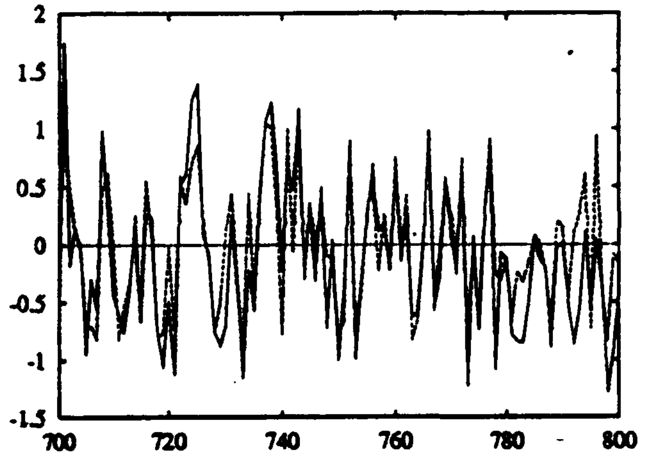


Figure 5. Identification example 2, Output # 2: Comparison between actual model output and network output.

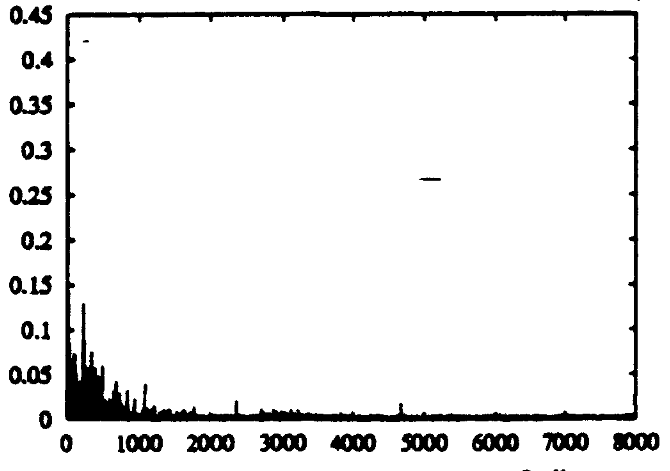


Figure 6. Prediction example: Online training error (squared).

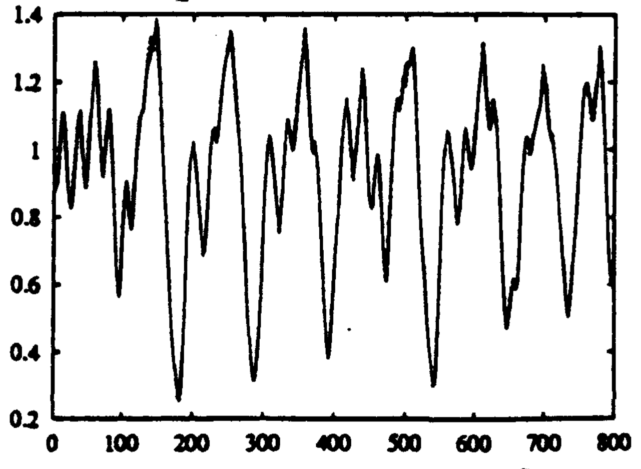


Figure 7. Prediction example: Comparison between actual series and six-steps ahead predicted series.

ANNIE - 91

Multi-layered Neural Networks for Arbitrary Approximation: An Explanation and Simulations

Girish Govind and P.A. Ramamoorthy
Mail Location # 30
Signal Processing and Computer Vision Group
Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, OH 45221-0030

Abstract

Recently there has been a lot of research into the approximation properties of multi-layered neural networks. The apparent ability of universal approximation by multi-layered neural networks is of interest as it would be very useful in several applications including system identification and control. In this paper we focus our attention on system identification and bring out certain similarities and differences between the conventional Volterra Series techniques for nonlinear system identification, and the neural network approach. Using concepts such as Continued Fraction expansions, the neural network is shown to be able to approximate very highly nonlinear systems with a few weights as compared to a Volterra Series which required a large number of coefficients to perform the same task. Also, the neural network can approximate nonlinear systems that cannot be approximated by the Volterra Series approach. Simulations are presented to show the superior performance.

1 Introduction

System Identification involves the selection of proper models and inferring the model parameters from observations. The quality of fit is obtained using a suitable error criterion. The model can be linear or nonlinear. Linear models are well established for system identification[1], however nonlinear system identification has not received as much attention as it is usually more difficult to come up with the proper models and algorithms to estimate their parameters. Hence, nonlinear systems are normally approximated using linear models by restricting the range of perturbation to a fixed and small range. However, such a model is restricted for system operations within that range, and to develop a globally valid model (valid for all inputs) for a nonlinear system, a nonlinear model has to be used.

The Volterra-Wiener approach is the most commonly used technique for nonlinear system modeling and is well established[2]. Neural Networks were proposed for nonlinear signal processing and system modeling about two years ago by Lapedes and Farber[3], and independently by us[4]. Since then there have been several papers applying the neural networks like a "black box" to almost every nonlinear task. However little attempt has been made to try to compare the neural network method with conventional methods of nonlinear system identification to put them in a proper perspective. Such an approach would enable us to identify the potentials and drawbacks of neural networks. Further, it will enable us to arrive at new adaptation algorithms and architectures.

2 Nonlinear System Identification

There are a number of Nonlinear System Identification techniques available in the literature (see for example, the excellent survey in [5]). These methods can be classified as - block-oriented systems, parameter estimation methods for particular model structures, heuristic GMDH (Group Method of Data Handling) methods, functional series methods of Volterra and Wiener, and the new multi-layered Neural Network approach [3, 4]. In this paper we concentrate on the two remaining methods, the Volterra series and the multi-layered neural network approach - both are general methods that can be used for identification of a large class of nonlinear systems. To relate to the rest of the paper, we now present the salient ideas of the two methods.

2.1 Volterra series

The Volterra series can be considered as a Taylor series with memory and can be written as

$$\begin{aligned} y[n] &= h_0 + \sum_{i=0}^{N} h_1[i]x[n-i] + \\ & \sum_{i=0}^{N}\sum_{j=0}^{N} h_2[i,j]x[n-i]x[n-j] + \cdots \end{aligned}$$

for the representation of a nonlinear time-invariant causal system. Just as a linear time-invariant system is completely characterized by its impulse response, a nonlinear system is represented (globally) by a Volterra series is completely characterized by its Volterra kernels – $h_1, h_2 \cdots$. Symmetry can be assumed in the higher order kernels e.g. $h_2[i,j] = h_2[j,i]$ without imposing any restrictions on the class of systems that can be represented.

According to the theory, any nonexplosive, time-invariant, causal system with a fading memory can be approximated by a Volterra Series[6]. In practice, the order cannot be increased to an arbitrarily high number as 1) then the measurement of each Volterra kernel is not simple, their individual responses cannot be separated and 2) the number of points in the kernel to be estimated increase exponentially with the order of the kernel and the delay (N) used. For a Volterra model using N inputs, for each order k, the number of points for which the kernel needs to be estimated is N^k/k. These problems with the Volterra series, impose severe restrictions on the application of Volterra series to many practical systems. Hence, most applications are restricted to systems where 'mild' polynomic nonlinearities are present and second or third order models are sufficient. Recursive least squares algorithms have been developed[7] and used in this paper for obtaining the second order and third order Volterra kernels with somewhat reduced computational complexity.

In addition, as the Volterra series can be considered as a power series with memory, there are other limitations. For example, it does not converge for certain nonlinear systems[2] such as saturating elements, and systems which have high order nonlinearities require a very large number of terms to yield an acceptable representation.

Wiener functionals alleviate the problem of measurement by using orthogonal functionals derived from the Volterra functionals for white Gaussian inputs, and are convergent for a broader class of systems. However, the problem of large number of terms in the functionals is still present and the input has to be white Gaussian (or made so) and hence is not very practical.

2.2 Multi-layered Neural Networks

The multi-layered feedforward net with sigmoidal nonlinearities was shown to be able to approximate arbitrary nonlinear systems by Lapedes [3] and independently by us [4]. Besides their approximation capabilities, multi-layered neural networks can be trained from a large data set without requiring any apriori information about the system (structure and order). The backpropagation algorithm is commonly used to train a multi-layered feedforward neural network[8].

The Kolmogrov Theorem is commonly used to justify the neural network approach to nonlinear system identification. This theorem states that a two layer neural network is sufficient for approximating arbitrary nonlinear systems given sufficient number of hidden nodes. Recently, it has been discussed that the Kolmogrov Theorem is actually irrelevant to the approximation property of neural networks [9] as Kolmogrov Theorem only talks about the representation of functions of a certain number of variables as functions (g_q) of functions (h_{pq} which can be sigmoidal) of fewer (one or more) variables. It is suggested that Kolmogrov Theorem may not apply to the representation properties of neural networks as it is based on using the same sigmoidal function in the place of properly chosen continuous functions. Certain approximations might require highly nonsmooth functions whereas the sigmoidal function is a very smooth function[9]. The smoothness property is very important for learning and generalization.

In spite of the above remarks, in practice the neural network method for approximating an arbitrary input-output map seems to work well though it is quite conceivable that there are specific methods that work better for a particular problem. In the next sections, we view the approximation problem from a different perspective and present arguments and simulation results that establish the validity for the use of multi-layered feedforward neural networks for modeling nonlinear systems.

3 Approximation by Neural Nets

Here we shall concern ourselves only with functions realized by the neural network after convergence, and not on the training. In such a network the sigmoidal function is the nonlinearity that imparts it the nonlinear mapping property. Other approximation schemes can also be depicted as networks having different basic functions. In general, the approximation of functions can be considered as a two-step process

- Choosing a set of basis functions $\Phi(x, w)$ that are defined in the region of interest of the input x, and
- Determining the parameters w in the basis functions to get a good fit.

In conventional approximation techniques the choice of basis functions is made considering the classes of functions that can be effectively approximated by them. To measure the quality of the approximation a distance measure (which is usually the L_2 norm) is used. Clearly, Volterra Series can also be considered as an approximation function where the basis functions consist of power and product terms in the inputs. However, approximation by multi-layered feedforward neural networks does not fit into the above picture of approximation theory. The approximation function $f(x)$ for the net is the nested sigmoid equation.

$$f(x) = g(\sum_i w2O_i g(\sum_j w12_j g(\sum_k wI1_k x_k)))$$

where the nonlinearity g is sigmoidal. Thus the basic function g is embedded in the network and so are the weights (variable parameters). Now we give methods of analyzing the above approximation functions using continued fraction expansions for the embedded sigmoid functions.

Continued fractions[10] have been used in the past for digital filter synthesis, computer evaluation of functions, coding theory and prime factorization algorithms. They yield representations for functions which are very exact. The expansion for tanh (a sigmoidal function) when truncated after the fourth convergent can be written as

$$tanh(x) \approx \frac{x}{1+\frac{x^2}{3+\frac{x^2}{5+\frac{x^2}{7}}}}$$

or

$$tanh(x) \simeq \frac{105x + 10x^3}{105 + 45x^2 + x^4}$$

A plot of this would show clearly that the truncated continued fraction expansion is very accurate while a corresponding truncated Taylor series expansion is wildly oscillating. In fact, some truncated forms of the expansion using Taylor series were even unbounded. Hence we will be using the "truncated continued fraction" expansion for the tanh function in this paper.

Consider the structure in Figure 1 with two hidden nodes and one output node. The input to the hidden layer node i is an affine transformation of the inputs

$$x1[i] = wI1[i][1].x(n) + wI1[i][2].x(n-1) + \theta1[i]$$

Then the output of each of the two nodes of the first hidden layer would consist of a ratio of polynomials in $x[n]$ and $x[n-1]$ with their cross-product terms. The numerator expression will have 10 terms (and coefficients) and the denominator expression having 14 terms (and coefficients). These terms are derived from the inputs and all the coefficients here depend on the values of the three free variables, the weights - $wI1[i][1]$, $wI1[i][2]$, and $\theta1[i]$. Thus there are only three "degrees of freedom" in choosing the 24 coefficients.

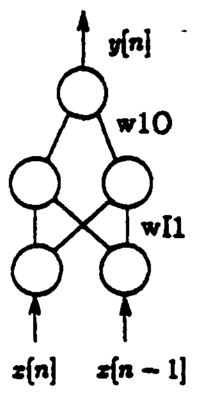


Figure 1: A simple neural network

It can be noted though that this form is clearly more general than that of the Volterra representation which takes the form of a power series. Further, in Volterra representation we make apriori assumptions about the nonlinear terms that need to be present whereas an infinite order is used in neural networks and can model much higher order nonlinearities that are practically impossible with the Volterra series.

At the next layer of weights the process is repeated and the output of the next layer is again a ratio of series in its inputs. It cannot be written out here conveniently but it can be visualized that the final form from input to output is a ratio of two infinite order series in the inputs.

It is known that for an arbitrary input-output map, an exact representation is impossible[11]. However as shown above, the Neural Networks can generally provide a better approximation than other techniques. Increasing the number of nodes in a layer; or increasing the number of layers increases the number of "degrees of freedom" for the neural network leading to better approximations though that also increases the possibility that the network may be trained on outliers in the case of noisy data.

When only one hidden layer is used, the output will consist of summations of sigmoidal outputs of affinely transformed inputs. The approximation capability of such a network would be very limited. Obviously adding more layers produces more complex nonlinear terms and hence intuitively it can be stated that a two-layer network would approximate an arbitrary mapping much better than a single-layer network with the same degree of freedom. The same conclusion is arrived in [12] using rigorous real analysis.

4 Simulations

In this section a number of simulation examples are provided to show the approximation capabilities of neural networks and compared with the performance of conventional Volterra nonlinear techniques.

In the simulations, the Normalized Root Mean Square Error (NRMSE)

$$NRMSE = \frac{E[(\hat{y}[n] - y[n])^2]^{1/2}}{E[(y[n] - E[y[n]]^2]^{1/2}}$$

is used as a performance index. Table 1 at the end gives the details of the simulations performed. In cases where the output values were scaled, the scaled data was used for all three simulations. Neural structures used have been specified as a-b-c-d where a, b, c, and d are the nodes in the input, first hidden, second hidden, and the output layers respectively.

Model 1

This is a purely linear model and was simulated to show that the neural network is also capable of approximating linear mappings. It should be noted that though a linear model would converge exactly to the simulated model and hence the fixed weight modeling error would be nearly zero (within a few iterations), the neural network error is only small (after several hundred iterations). Thus a linear model should be tried before modeling with neural networks as for linear mappings the linear model is more accurate, and less effort is required for a fit.

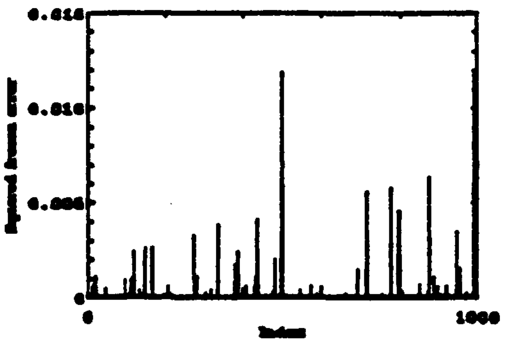


Figure 2: Model 1 Neural frozen error

Model 2

A second order Volterra model was very accurate and estimates of the parameters were exact. The multi-layered neural network with 3-5-5-1 nodes was as exact as the Volterra model. Thus for structured nonlinearities, the neural network is capable of working as well as Volterra models.

Model 3

This simulation clearly shows the ability of a neural network to model high order nonlinearities (the sine function can be expanded as an infinite Taylor series about x=0). The Volterra model does not converge. On the other hand, a multi-layered neural structure of 2-5-5-1 nodes was very accurate.

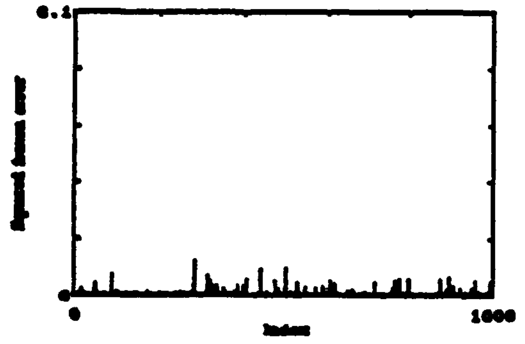


Figure 3: Model 2 Volterra frozen error

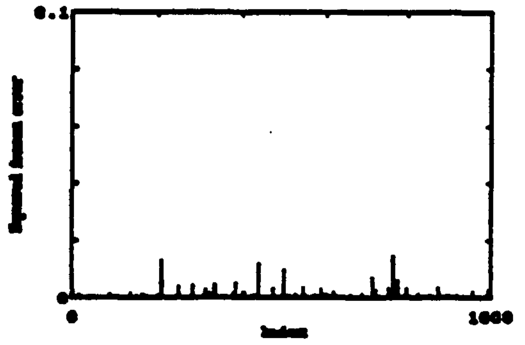


Figure 4: Model 2 Neural frozen error

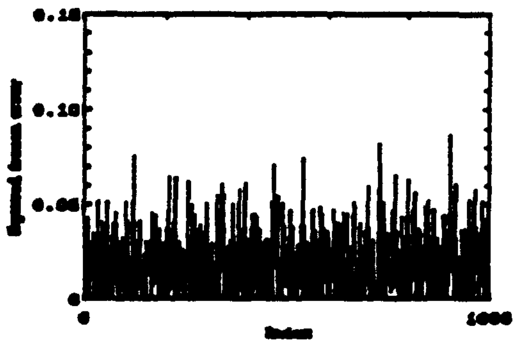


Figure 5: Model 3 Volterra frozen error

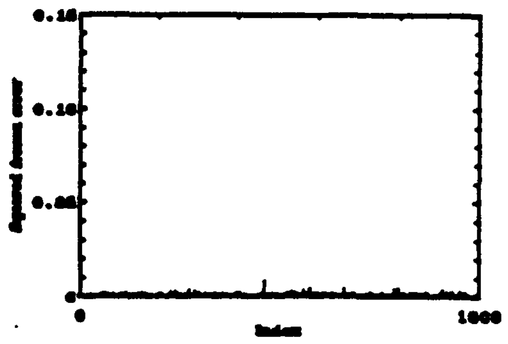


Figure 6: Model 3 Neural frozen error

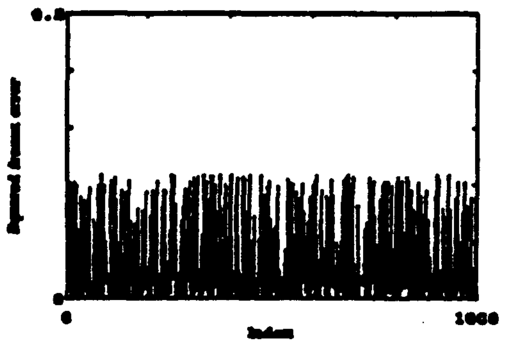


Figure 7: Model 4 Volterra frozen error

Model 4

Even though the nonlinearity was simple, nonexplosive, and time-invariant, no suitable Volterra model could be found. A simple neural network of 1-2-2-1 nodes was used and gave a very accurate representation of this nonlinearity. The graph is not shown here as the error was insignificant in comparison with the error with the Volterra model.

The frozen input-output map is shown in Figure 8. The neural network modeled the nonlinearity almost exactly and is indistinguishable from the original input-output map. For this nonlinear system there are two Volterra models, depending on the range of the input, one model for $| x | < 0.2$ and another one for $| x | > 0.2$. Modeled separately it would give accurate Volterra models but this may not be possible as the input to the nonlinearity could be an internal signal in a system. The neural network has no such problem as it has a numerator and denominator of polynomials representation.

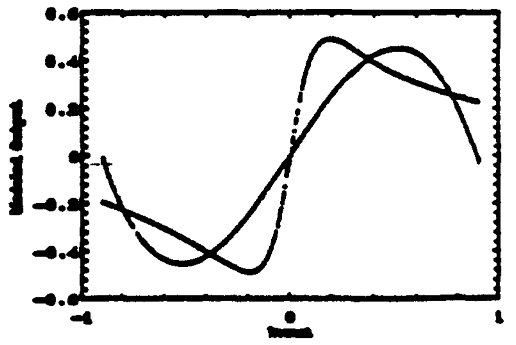


Figure 8: Model 4 approximations

Simulated System	Input	Output	Simulations		
			Model	Figure	NRMSE
$y[n] = 0.5u[n] + 0.25u[n-2]$	White Gaussian $N(0.0, 0.36)$ $u[n], u[n-1]$ & $u[n-2]$	$[-1.38, 1.22]$ scaled to $[0.05, 0.95]$	Volterra 1st order	—	0.0001
			Neural 3-5-5-1	3	0.1026
$y[n] = 0.3u[n] + 0.25u[n-1]u[n-2]$	White Gaussian $N(0.0, 0.36)$ $u[n], u[n-1]$ & $u[n-2]$	$[-0.92, 0.72]$ scaled to $[0.05, 0.95]$	Volterra 2nd order	4	0.0645
			Neural 3-5-5-1	5	0.0215
$y[n] = 0.7\sin(u[n]) + 0.2u[n-1]$	White Uniform $[-4.0, 4.0]$ $u[n]$ & $u[n-1]$	$[-1.50, 1.48]$ scaled to $[0.05, 0.95]$	Volterra 2nd order	6	0.6742
			Neural 2-5-5-1	7	0.1083
$y[n] = \dfrac{5x[n]}{1 + (5x[n])^2}$	White Uniform $[-0.9, 0.9]$ $u[n]$	$[-0.5, 0.5]$,	Volterra 3rd order	8	0.4304
			Neural 1-2-2-1	—	0.0269

Table 1. Simulations on various models.

5 Conclusions

In this paper multi-layered neural networks using "tanh" building blocks are compared with conventional Volterra series methods for the task of nonlinear system identification. For the neural model, model selection corresponds to selecting a suitable structure for the representation which controls the number of degrees of freedom. For highly nonlinear systems, multi-layered neural networks provide a representation with only a few weights, where conventional Volterra series methods would not be computationally practical. Also, for some simple nonlinear systems which are bounded-input and bounded-output, the Volterra series does not exist although it is handled easily by a small network. Finally, simulations were provided which support the ideas presented here.

References

[1] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[2] M. Schetzen, "Nonlinear system modeling based on the Wiener theory," *Proc. of the IEEE*, vol. 69, pp. 1557-1573, December 1981.

[3] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: Prediction and system modeling," tech. rep., Los Alamos National Laboratory, 1987.

[4] P. A. Ramamoorthy, G. Govind, and V. Iyer, "Signal modeling and prediction using neural networks," in *International Neural Network Society First Annual Meeting*, (Boston, MA), September 1988.

[5] S. A. Billings, "Identification of nonlinear systems - a survey," *IEE Proc. Part D*, vol. 127, pp. 272–285, November 1980.

[6] S. Boyd and L. O. Chua, "Fading memory and the problem of approximating nonlinear operators with Volterra series," *IEEE Trans. on Circuits and Systems*, vol. CAS-32, pp. 1150–1161, November 1985.

[7] V. J. Mathews and J. Lee, "A fast recursive least-squares second order Volterra filter," in *Proceedings of the ICASSP*, 1988.

[8] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4–22, April 1987.

[9] F. Girosi and T. Poggio, "Representation properties of networks: Kolmogrov's theorem is irrelevant," *Neural Computation*, vol. 1, pp. 465–469, 1989.

[10] M. H. Hayes, "Continued fractions," *IEEE ASSP Magazine*, July & October 1989.

[11] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," tech. rep., M.I.T. AI Laboratory and Center for Biological Information Processing, 1989.

[12] G. Cybenko, "Continuous valued neural networks with two hidden layers are sufficient," *Mathematics of Controls, Signals and Systems (submitted)*, 1988.

From: International Conference on Systems Engineering, Pittsburgh, August 199

Multi-layered Neural Networks and Volterra Series: The Missing Link

Girish Govind and P.A. Ramamoorthy
M. L. 30, Signal Processing and Computer Vision Group
Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, OH 45221-0030

Abstract

This paper is an attempt to bring out the similarities and differences between the conventional Volterra series techniques and the new neural network approach. The analysis is done from the point of view of representation capabilities for nonlinear systems and it is shown that a small neural network can represent high order nonlinear systems while a very large number of terms are required for an equivalent Volterra series representation. This is shown by means of a series expansion of a neural network. Finally this paper analyzes issues common to the two nonlinear modeling approaches.

1 Introduction

Many problems in controls and signal processing require accurate models of the systems involved, systems which are usually nonlinear to some extent. System identification techniques are well established for linear systems and are widely used, but methods for nonlinear systems have not received as much exposure. This is due to their inherent complexity, and difficulties in deriving identification algorithms for models which would be applicable to a large class of nonlinear systems. Although it is possible to represent nonlinear systems by linear models for a restricted operating range and use the well developed system identification techniques [1], a nonlinear process can only be characterized by a nonlinear model.

Nonlinear system identification [2] methods can be of several types – functional series methods of Volterra and Wiener, block-oriented systems, parameter estimation methods for particular model structures, heuristic GMDH (Group Method of Data Handling) methods, and the new Neural Network approach [3]. Block oriented techniques require structure detection, and parameter estimation methods vary depending on the model used. The GMDH algorithms are self-organizing and heuristic in nature.

In this paper we concentrate on two of these methods, the Volterra series and the multi-layered neural network approach – both are general methods that can be used for identification of a large class of nonlinear systems. In the next two sections we present the salient points of the two methods.

2 Volterra series

The Volterra series can be considered as a Taylor series with memory and can be written as

$$\begin{aligned} y[n] &= h_0 + \sum_{i=0}^{N} h_1[i]x[n-i] + \\ &\sum_{i=0}^{N}\sum_{j=0}^{N} h_2[i,j]x[n-i]x[n-j] + \cdots \end{aligned}$$

for the representation of a causal system. A nonlinear system represented (globally) by a Volterra series is completely characterized by its Volterra kernels.

As the Volterra series can be considered as a power series with memory, there are limitations of convergence in its application to nonlinear problems [4]. For instance, the Volterra series cannot be used to represent saturating elements, and systems which have high order nonlinearities require a very large number of terms to yield an acceptable representation. However according to the Weierstrass theorem the system can be uniformly approximated over a bounded interval. Measurement of the Volterra kernel is not easy as the contribution of each Volterra operator cannot be separated from the total system response.

As can be seen from the above series, the number of terms in the kernels of the series increases exponentially with the order of the kernel and the delay (N) used. This is the most difficult problem with the Volterra series and imposes severe restrictions on the application of Volterra series to many practical systems. Hence, most applications are restricted to using second order models. Recursive least squares algorithms are used in this paper for obtaining the second order and third order Volterra kernels with reduced computational complexity.

Wiener functionals alleviate the problem of measurement by using orthogonal functionals derived from the Volterra functionals for white Gaussian inputs, and are convergent for a broader class of systems. However, the problem of number of terms in the functionals is still present and the input has to be white Gaussian (or made so) and hence is not very practical.

3 Multi-layered Neural Nets

The error propagation net was first shown to be able to approximate arbitrary nonlinear systems by Lapedes [3] and independently by us [5]. The error propagation network is explained in detail in [6].

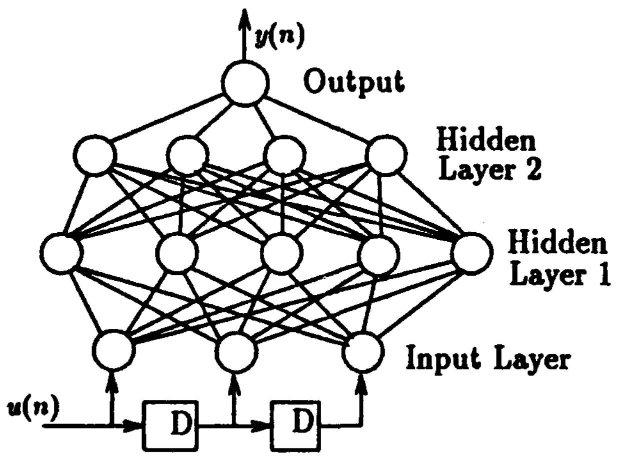


Figure 1: Multi-layered Neural Network

The neural network approach to Nonlinear system identification is based on the Kolmogrov Neural Network existence theorem [6] which states that a two layer neural network is sufficient for approximating arbitrary nonlinear systems given sufficient number of hidden nodes. Although the Backpropagation algorithm which is used to train the multilayered neural networks is a straightforward extension to the well-known Least Mean Square (LMS) algorithm for linear structures, its convergence parameters have to be finely adjusted to get fast convergence. In the simulations in this paper we have used a conjugate-gradient technique for training the weights – this provides relatively fast training of weights and no adjusting of step and momentum terms is required to obtain convergence.

4 Expansions for Neural Nets

In this paper we shall concern ourselves with functions realized by the neural network after convergence. The nonlinearity used in multi-layered neural networks is known, typically the sigmoid or the tanh function. In this paper we shall use the tanh function for the expansions, simulations and the discussions. Corresponding work for the sigmoid can be easily seen as the two functions are related. The tanh function can be written as

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

and the exponentiation function can be expanded as a Taylor series about $x = 0$.

$$e^{2x} = 1 + 2x + \frac{(2x)^2}{2!} + \frac{(2x)^3}{3!} + \cdots$$

which is a valid expansion for all real values of x. Therefore the *tanh* function can be written as

$$tanh(x) = \frac{2x + \frac{(2x)^2}{2!} + \frac{(2x)^3}{3!} + \cdots}{2 + 2x + \frac{(2x)^2}{2!} + \frac{(2x)^3}{3!} + \cdots}$$

for all values of x.

Cybenko [7] and others have shown the approximation capabilities of the multi-layered structure using real analysis techniques. This paper analyses the multi-layered structure using series expansions to show the origin of some of those properties and relate them to corresponding ones of the Volterra series at a more intuitive level.

Consider the structure in Figure 2 with two hidden nodes and one output node. The input to the hidden layer node i is

$$x1[i] = wl1[i][1].x(n) + wl1[i][2].x(n-1) + theta1[i]$$

and if we assume the representation of *tanh* to be approximated by terms in the expansion up to the third order, then the output of each of the two nodes of the first layer consists of a ratio of third order polynomials *poly*[*i*] each having 10 coefficients multiplying terms derived from the inputs. These 10 coefficients are in turn derived from the three "degrees of freedom" $wl1[i][1]$, $wl1[i][2]$ and $theta1[i]$ and the output is of the form

$$\frac{poly[i]}{2 + poly[i]}$$

The polynomial *poly*[*i*] can thus be chosen only with a certain number of degrees of freedom and

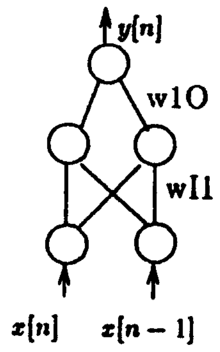


Figure 2: A simple neural network

this restricts its use to only very simple systems. This form is however clearly more general than that of the Volterra series which only consists of a series as this expansion has a numerator series and a denominator series. Even though here only a truncated third order form is shown it is actually an infinite order polynomial and can model much higher order nonlinearities than are practically possible with the Volterra series.

At the next layer of weights the process is repeated and the output of the next layer is a ratio of polynomial in its inputs. It cannot be written out here conveniently but it can be visualized that the final form from input to output is a ratio of two polynomials in the inputs. Thus

$$y[n] = \frac{\text{Numerator polynomial in inputs}}{\text{Denominator polynomial in inputs}}$$

and both polynomials are of an infinite order. There are however only limited "degrees of freedom" in choosing these polynomials as their coefficients are a function of the weights and in the network above there are only 9 weights. More hidden nodes may be required to tailor the response more closely.

5 Simulations

In this section the insight derived in the last section is strengthened by simulation examples.

Model 1

$$y[n] = \frac{5 \times x[n]}{1 + (5 \times x[n])^2}$$

This model is from [4] and here $x[n]$ was a uniformly distributed random variable with excursion [-0.9, 0.9]. A third-order no-memory (N = 0) Volterra model was used and the four terms were identified using Recursive Least Squares. A neural network structure of 1-2-2-1 nodes was used. Modeling error results with frozen weights are shown in Figure 3a and 3b. Figure 3c is a plot of $y[n]$ vs. $x[n]$ with frozen parameters and the neural network modeled the nonlinearity almost perfectly while the third-order Volterra model yielded a very poor approximation.

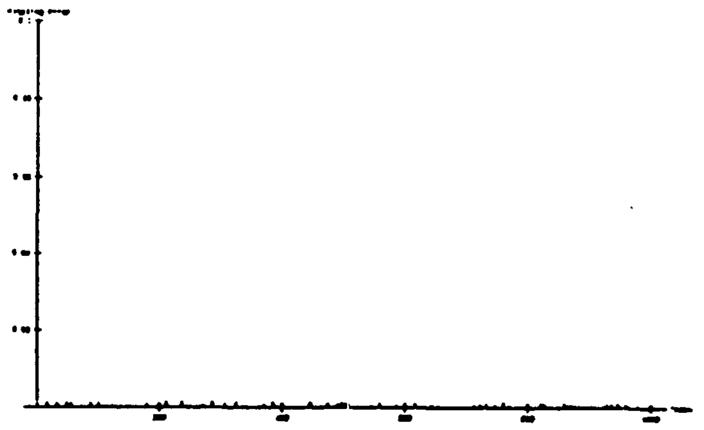


Figure 3a. Neural modeling error

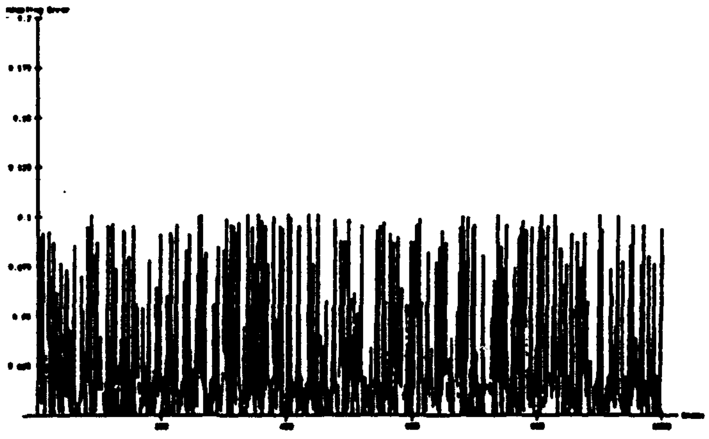


Figure 3b. Volterra modeling error

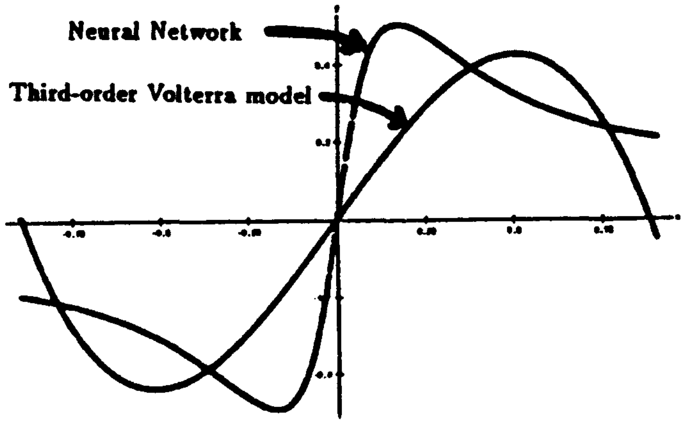


Figure 3c. Modeled nonlinearity

Model 2

$$y[n] = \begin{cases} x[n] * x[n-1] & \text{if } x[n] > 0 \\ 0.4 \times x[n] + 0.9 \times x[n-1] & \text{otherwise} \end{cases}$$

which is a threshold model. Here $x[n]$ was a uniformly distributed random variable with excursion [-0.9, 0.9] but the desired output for both the neural network and the Volterra simulation was scaled down to lie between [-0.9,0.9] (within the range of tanh). The neural network had a 2-5-5-1 structure and Figure 4 shows the frozen error after convergence for the two simulations.

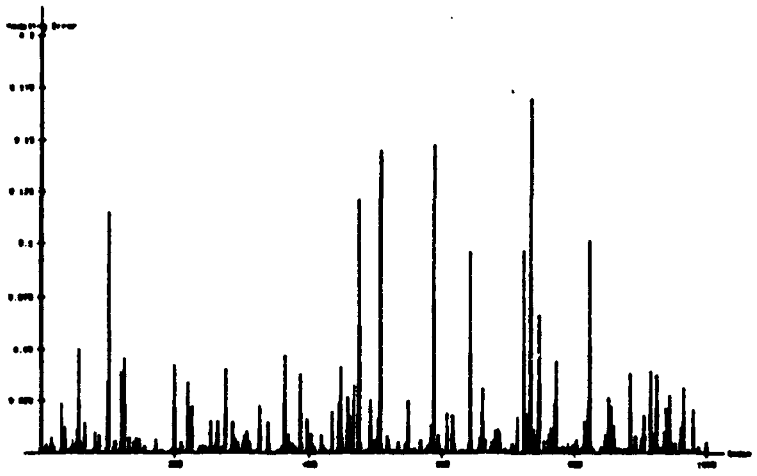


Figure 4a. Neural modeling error

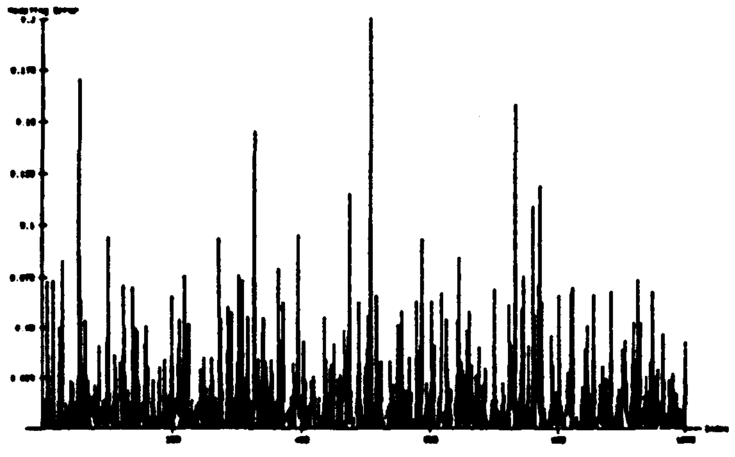


Figure 4b. Volterra modeling error

6 Discussion

The first model considered does not have a Volterra series representation if the range of $x[n]$ is more than (-0.2, 0.2), which is the case here. The second model considered also does not have a representation in Volterra series and the neural network being a more general model as was shown by the expansions, models this system with a much lower modeling error.

A point to note here is that the structure of the model being simulated was not used in the neural network simulation as there is no known way to use apriori information for choosing the network weights. Also, the error achieved after freezing the weights is not the lower bound, the adaptation was stopped when the error level reached an acceptable low value. The neural network model parameters (weights) are few as compared to the number of terms in the numerator and denominator series, hence there are several constraints on the values of the weights and training of these networks requires fine adjustment of the learning parameters.

7 Conclusions

In this paper we have shown how the multi-layered neural network is able to approximate arbitrary nonlinear systems while the Volterra series models cannot give a representation for these simple bounded-input, bounded-output models. This has been shown by analyzing the form of the nonlinearity and using its expansion to compare with those of the Volterra series. Finally, simple models are simulated to support the theory.

References

[1] L. Ljung, *System Identification: Theory for the user*. Englewood Cliffs, New Jersey: Prentice Hall, 1987.

[2] S. Billings, "Identification of nonlinear systems – a survey," *Proc. IEE, Pt. D*, vol. 127, pp. 272–285, November 1980.

[3] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: Prediction and system modeling," tech. rep., Los Alamos National Laboratory, LA-UR-87-2662, 1987.

[4] M. Schetzen, "Nonlinear system modeling based on the wiener theory," *Proc. IEEE*, vol. 69, pp. 1557–1573, December 1981.

[5] P. Ramamoorthy, G. Govind, and V. Iyer, "Signal modeling and prediction using neural networks," in *INNS First Annual Meeting, Boston*, September 1988.

[6] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4–22, April 1987.

[7] G. Cybenko, "Approximation by superpositions of a sigmoidal function," tech. rep., Unpublished manuscript, 1988.

IFSICC-92

ADAPTIVE FUZZY EXPERT SYSTEMS FOR CONTROL APPLICATIONS

P.A. Ramamoorthy, Shi Zhang and Song Huang

Department of Electrical and Computer Engineering, University of Cincinnati, M. L. 30, Cincinnati, OH 45221-0030, USA

ABSTRACT. A fuzzy expert system (FES) in general performs the task of functional mapping (that is, for a set of values of the inputs, the corresponding values of the outputs are produced), and hence can be used for system modeling or controller design. The functional mapping is achieved based on the six attributes of the system, namely, inputs, outputs, fuzzy sets of the inputs and outputs and their associated membership functions, fuzzy associative rules and the defuzzification procedure. The first four attributes can be defined rather easily based on some knowledge about the problem at hand. However, it is difficult to obtain the other attributes due to the non-availability of experts or the inability of the experts to represent their knowledge in a form needed by the expert system. Also, even in situations where the rules are available, they may not be optimal. Hence the need to develop adaptive techniques to optimize the performance of a fuzzy expert system. In this paper, we present a very general and systematic approach to this adaptation problem and discuss the advantages of such an approach. The problem of backing up a truck to loading dock from any reasonable location is used as an example and an optimal controller has been designed for this problem using this approach. The results obtained indicate that the approach is indeed viable and can be applied to many problems.

KEYWORDS Fuzzy logic; Fuzzy controller; Adaptive fuzzy systems; Fuzzy expert systems.

INTRODUCTION

Fuzzy expert systems can be considered as vehicles for functional mapping that maps an input u (a vector of size N) into an output y (a vector of size M) by the function f: $u \rightarrow y$. The functions f can be nonlinear. The mappings are obtained by splitting the inputs and outputs into smaller and overlapping intervals or fuzzy sets, assigning membership functions (that indicate the degree of belonging of a particular input or output value to the various fuzzy sets of those inputs or outputs), fuzzy rule base or fuzzy associative memories (FAMs) that define the output fuzzy sets to which outputs belong for inputs belonging to certain input fuzzy sets, and defuzzification procedure that leads to crisp output values from the selected output fuzzy sets and the membership functions. More details on fuzzy logic and fuzzy expert systems can be seen in references [1-5].

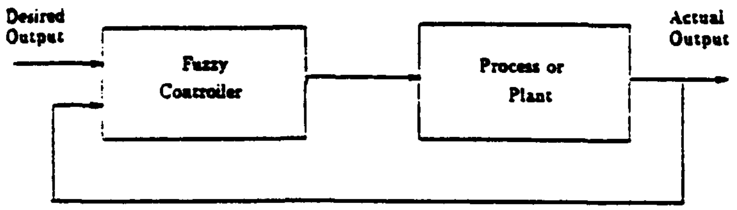


Figure 1. Fuzzy expert system as a controller for controlling a process.

Fuzzy expert systems can be excellent candidates as controllers for controlling machines and processes (Fig.1) as they enable implementation of complex mappings, especially nonlinear ones, through a systematic procedure involving linguistic representation of common-sense knowledge. Further, it allows a model free representation of the mapping and hence overcomes the limitations and the problems associated with model-based controllers.

A minimum requirement in the use of fuzzy expert systems is that the fuzzy rules are available. In many practical applications, it is possible to arrive at approximate rules [1] though such available rules may not lead to an optimal controller. In this paper, we propose the adaptation of the rules to arrive at an optimal controller. The approach is explained through a specific example defined in the following section.

PROBLEM DEFINITION

Here we consider the problem of designing an optimal controller to successfully back up a truck to a loading dock from any reasonable initial location. This is a typical controller design for a nonlinear plant. Nguyen and Widrow [6] recently showed that a nonlinear controller in the form of two-layer neural network architecture with 26 nodes can be successfully designed. We chose this problem as an example since the mapping is complex and involves a large number of starting points and trajectories. Kong and Kosko [7] considered the same problem and compared the performance of such a neural network based controller with that of a controller based on fuzzy expert system composed of 35 rules. They observed that even their simple FES leads to smoother trajectories than those produced by the two-layer network. We use their rule base as a starting point for our adoption procedure.

Fig. 2 shows the details of the truck-backer-upper problem and input and output variables of the system. Given enough clearance between the truck and the loading dock, the y-position can be omitted as an input to the controller. The ranges of the inputs, x-position, and the truck

[1]If this is not readily obvious, one should consider the tasks such as driving, operating a machine tool etc and the basis for their operation. We do such tasks sub-consciously or as-a-matter-of-fact. Once we master the techniques, we can put them into approximate rules with some effort.

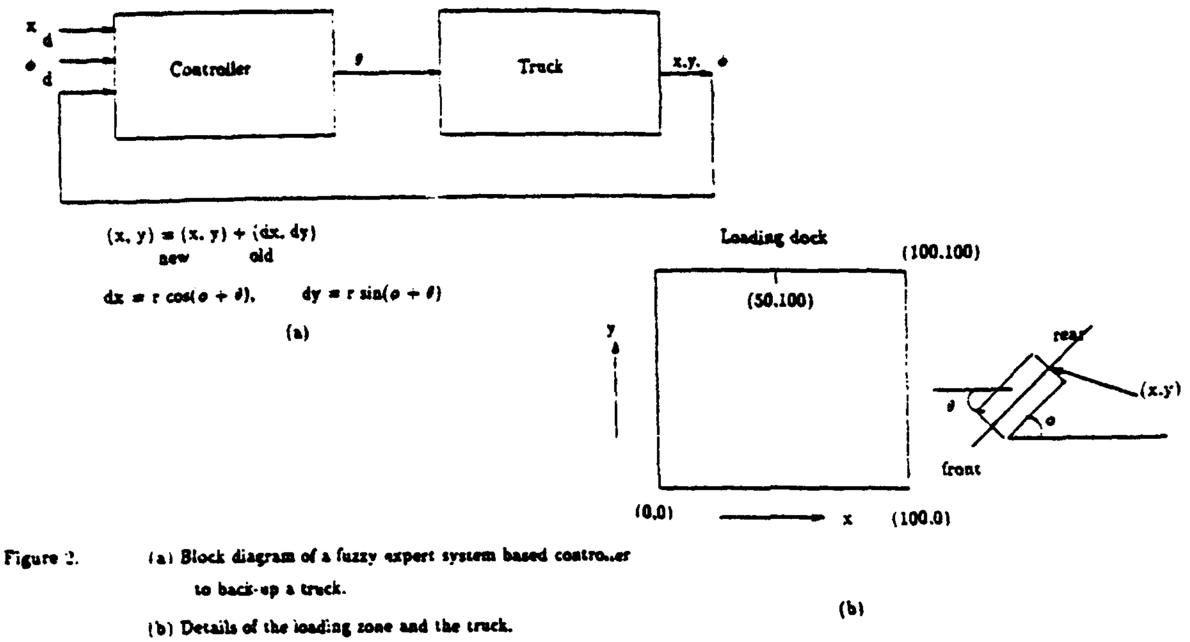


Figure 2. (a) Block diagram of a fuzzy expert system based controller to back-up a truck.
(b) Details of the loading zone and the truck.

orientation angle ϕ, and the output, steering signal θ, are given as:

$$\phi : [0, 360]; x : [0, 100]; \theta : [-30, 30]$$

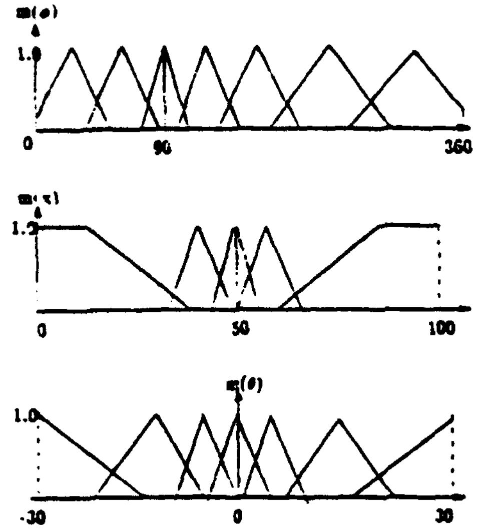


Figure 3. Membership functions of the various fuzzy sets of the input and the output variables.

		x-position				
		LE	LC	CE	RC	RI
angle (ϕ)	RB	PB	PB	PM	NB	NB
	RU	ZE	PS	PM	PB	PB
	RV	NB	NM	PS	PM	PB
	VE	NB	NM	ZE	PM	PB
	LV	NB	NM	NS	PM	PB
	LU	NB	NB	NM	NS	ZE
	LB	PB	PB	NM	NB	NB

Figure 4. The rulebase of the truck back-up problem.

Having identified the variables and their ranges, fuzzy subsets of the variables must be specified. Fuzzy subsets are simple linguistic terms and associated numerical values corresponding to the input and output variables. They are used to split the total range of the variables into smaller and overlapping ranges. The next step is the selection and identification of the membership functions associated with the various fuzzy sets. The membership functions simply describe the degree of association of a particular input or output value to the fuzzy subsets corresponding to that input or output variable. The membership functions of the IO variables for the truck-backer-upper problem are given in Fig. 3. Membership functions can have different shapes depending on the designer's preference, knowledge or experience. Triangular and trapezoidal shapes are used here for simpler calculation and better description of the problem.

A fuzzy rule base or fuzzy associative memory (FAM) is a collection of fuzzy rules which define or describe the relationship between the input fuzzy sets and output fuzzy sets. The rules are given in the conventional IF-THEN form, with an antecedent part that describe the conditions, and a consequent part that states the conclusions or actions. Fig.4 is the rule base for the truck-backer-upper controller problem as defined by Kong and Kosko. This rule base indicates the possible ranges for the output values given the input values. The final crisp output values of a fuzzy expert system are determined using a procedure known as the defuzzification process. Min and Max inference with centroid method is mostly used as the defuzzification procedure and truck-backer-upper control problem as well. The MIN-centroid procedure is illustrated in Fig. 5 and is carried out as follows:

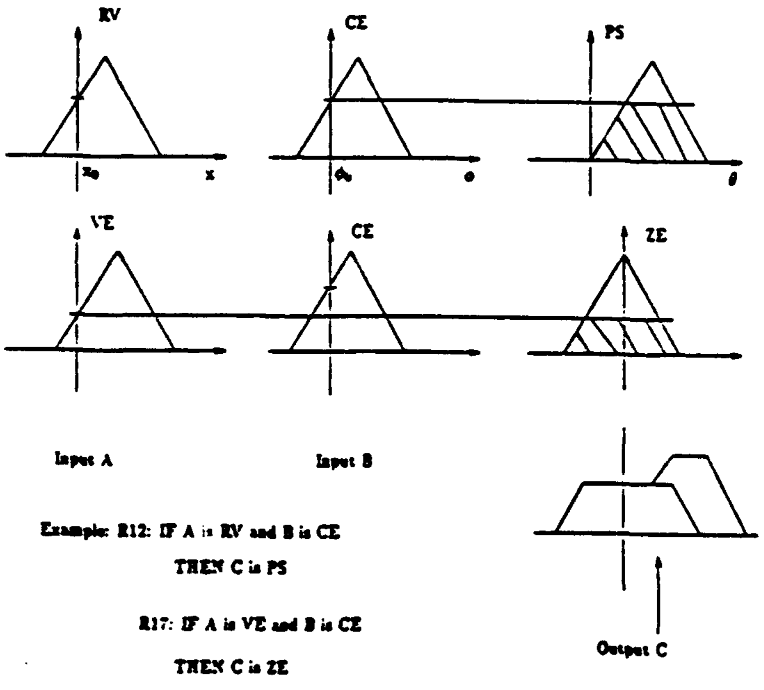


Figure 5. An example of MIN-Centroid procedure.

If the inputs x,ϕ are x−0 and ϕ_0 respectively, at some particular instant, x_0 can fall under the domain of a maximum of two fuzzy sets (assuming that the fuzzy sets are chosen to lead to a max of two fuzzy sets for any input value) and ϕ can similarly fall under the domain of a maximum of two fuzzy sets. Thus we have either one, two or four rules firing for any given inputs values. In the case of a single rule, if the rule is in the form of "AND" (OR), the values of the membership functions at $x = x_0$ and $\phi_0 = \phi_0$ are found and the minimum (maximum) is chosen. This value is used to clip the output membership function corresponding to the output fuzzy set selected by the rule and the centroid of the clipped area becomes the output θ_0 of the fuzzy system. If more than one rule gets activated, the combined centroid of the clipped area is the output. The trajectories of the truck from various initial positions backing up to the loading zone controlled by this fuzzy expert system is given in Fig. 6 (for clarity purposes, we have shown this as four separate diagrams corresponding to various starting positions). Though Kong and Kosko claimed that the trajectories produced by the FES are better than the ones produced by the neural net controller, it can be observed that the FES controlled trajectories are not smooth either.

ADAPTIVE FUZZY CONTROLLER

We have a fuzzy controller that succeeds in backing up the truck to the loading dock from any starting point. However, the performance is not very satisfactory and thus there is the need to consider adaptation of the fuzzy controller to obtain more smoother trajectories.

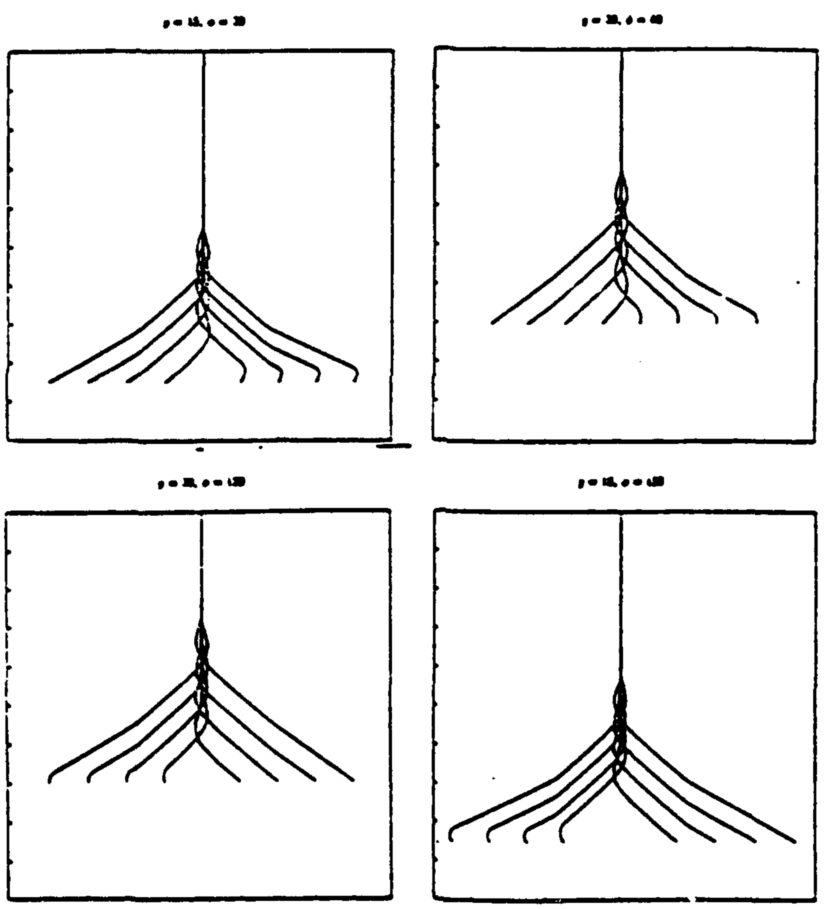


Fig. 6 Truck trajectories of the original controller

There are many ways that the fuzzy system or controller can be adapted to optimize the performance. In Fig. 7, we show the general concept. There we have, in addition to showing in block diagram of the steps involved in calculating the output of a fuzzy system, shown weights at strategic places which can be adapted or modified to obtain improved performance. We can also define different weights corresponding to each segment of the membership function with a constant slope and regions where the fuzzy sets overlap and so on (Fig. 7B). Such assignments would make the

Block diagram of a an adaptive MIN/CENTROID method of output calculation, where the weights have to be adjusted to obtain the required mapping. Steps shown are for one output calculation when J rules are invoked

ABBREVIATION

N - Number of inputs
M - Number of outputs
x_i - ith input
y_i - ith output
x_{i0} - Value of ith input at a particular time
y_{i0} - Value of ith output at a particular time
$MF_{kj}(y_i)$ - Membership function of fuzzy set kj of output y_i
$MVj_{x_i}(x_{i0})$ - Membership function value at $x_i = x_{i0}$ of the fuzzy set of input x_i that invokes rule j

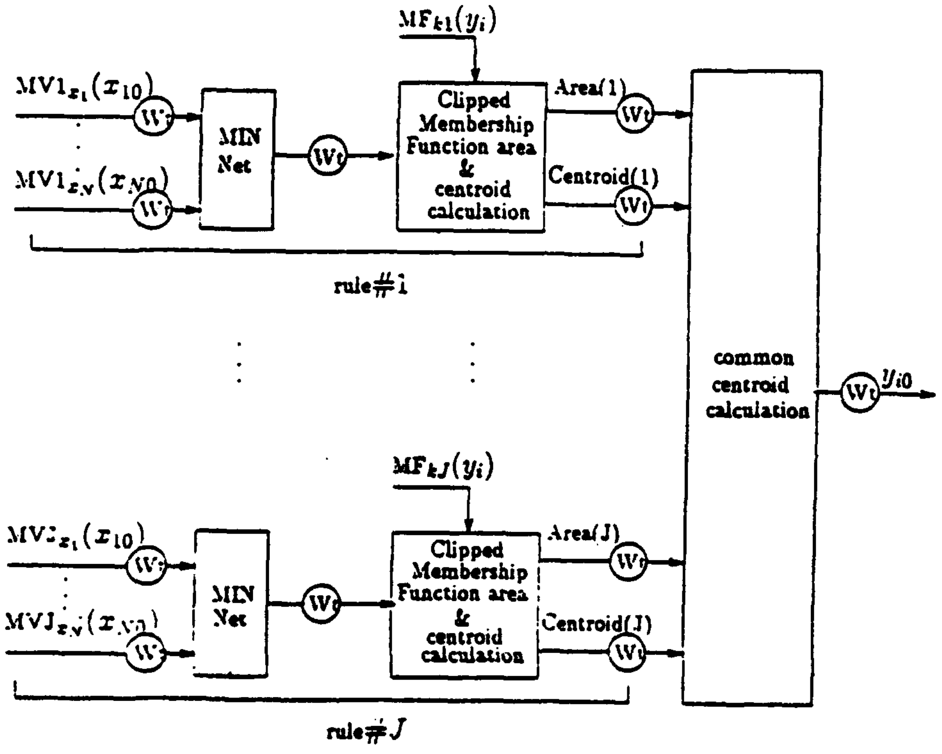


Fig. 7A

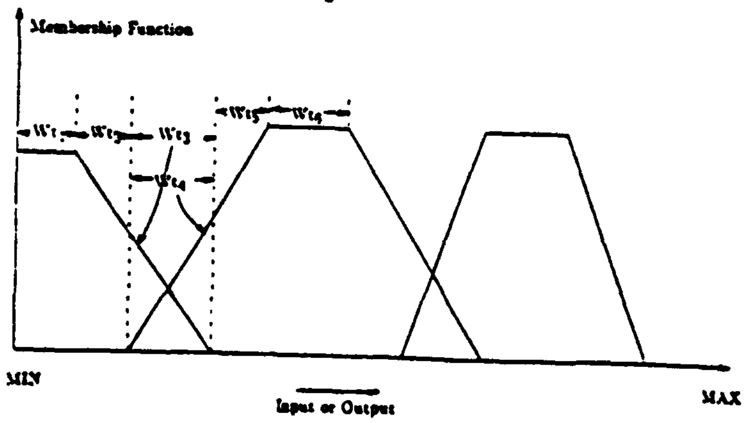


Fig. 7B. Separate weights defined for various fuzzy regions

adaptation simpler by making the individual adaptation domains smaller and lead to a technique that can be analyzed in a systematic manner. In this paper, we consider adaptation of the weight in the overlapping fuzzy regions only. The optimization is performed as follows:

We assume that the truck is at some random position $(x_{old}, y_{old}, \phi_{old})$ and the corresponding controller generated angle is θ_{old}. The truck moves by a fixed distance with a steering angle of $\theta_o ld$, and comes to a new position $x_{new}, y_{new}, \phi_{new}$, and the corresponding controller angle for the new position is denoted as θ_{new}. The objective function minimized is then given by:

$$J = (\theta_{new} - \theta_{old})^2$$

and leads to weight update equations:

$$\Delta w = \varepsilon[\theta_{new} - \theta_{old}]$$

$$w_{new} = w_{old} + \Delta w$$

where ε is a positive, very small constant. After the weights are adjusted using the above update equation, the truck is moved to a new random location and the process is repeated (we stopped the algorithm after 10,000 iterations). The results are shown in Fig. 8-12. In Fig. 8 we show the learning curve of two of these weights . Figs. 9-11 give a specific trajectory of the truck, the truck angle ϕ , and the steering angle (control signal) using the original and the optimized fuzzy controllers. Fig.12 shows the performance index that is being minimized. In Fig. 13, we show the trajectories starting from various initial positions. As can be seen from the figures, we are able to achieve substantial improvement in the performance by attaching weights to the rules and adapting them.

SUMMARY

The use of tunable fuzzy expert systems as nonlinear controllers in the control of a nonlinear plants is considered in this paper. We show how weights can be attached to fuzzy expert systems in a systematic manner and adapted to optimize a desired performance. Results from a specific application are shown and the results indicate that it is indeed possible to achieve good performance by tuning the controller. Of course, more work needs to be done along the lines indicated in this paper. We are currently working on those problems.

REFERENCE

[1] L.A. Zadeh, "Outline of a New approach to the Analysis of Complex Systems and Decision Processes," *IEEE Trans. SMC*, pp. 28-44, 1973.

[2] L.Z. Zedeh, "Commonsense Knowledge Representation Based on Fuzzy Logic," *IEEE Comp.* Vol. 10, pp. 61-65, 1983.

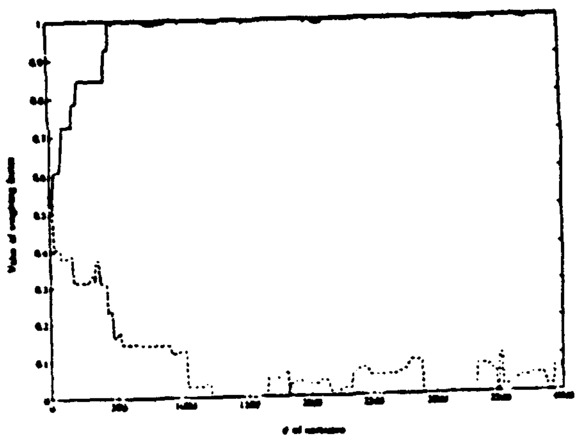


Fig. 8. Diagram of the learning course of two of the weights

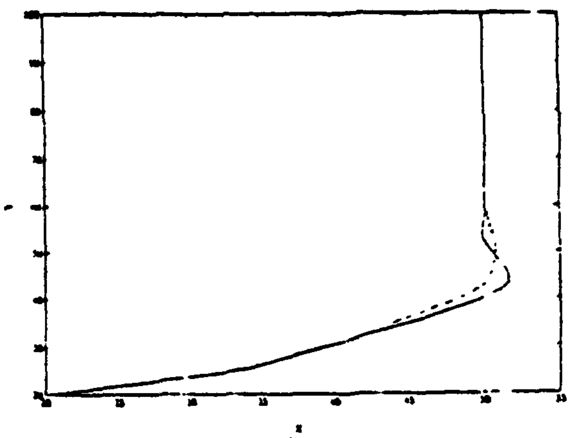


Fig. 9 Truck trajectories using the original controller and the optimized controller

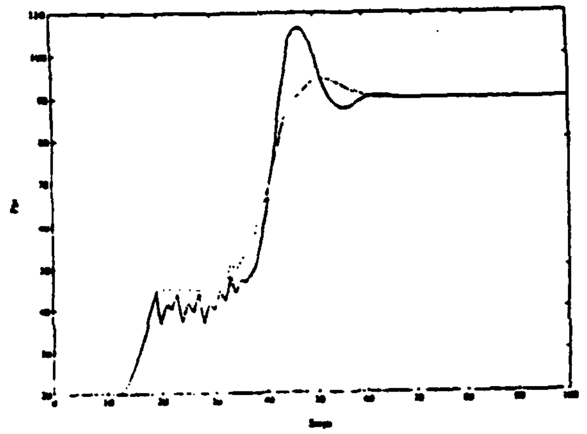


Fig. 10 Truck angles φ using the original controller and the optimized controller

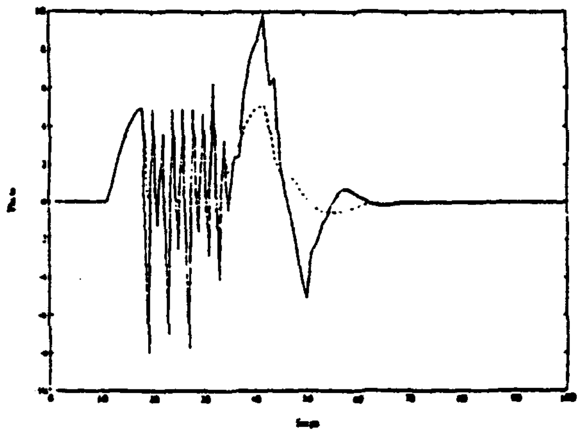


Fig. 11 Steering signals θ using the original controller and the optimized controller

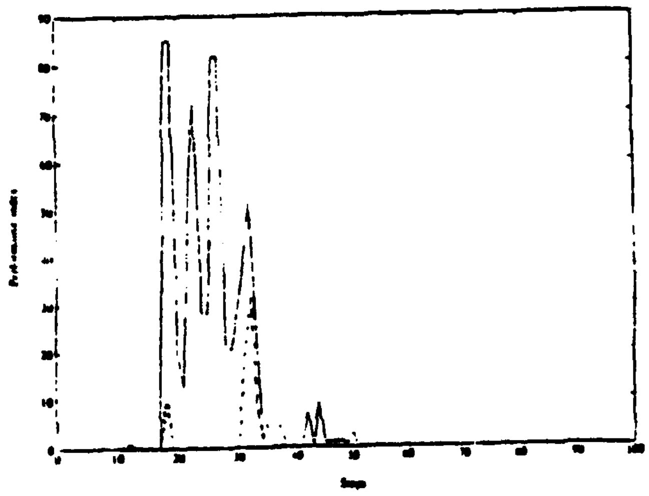


Fig. 12 Diagram of the performance values using the original controller and the optimized controller

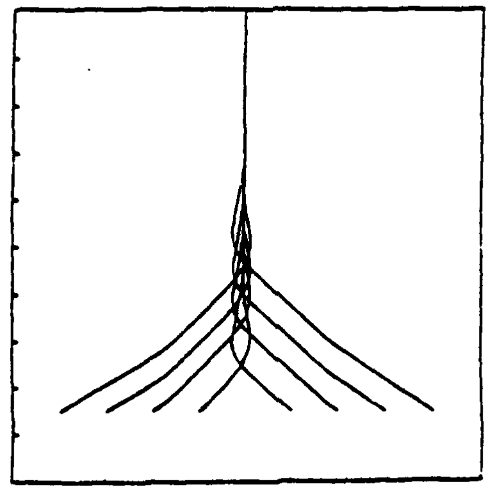


Fig. 13 Truck trajectories of the optimized controller

[3] Fuzzy Sets and Application:Selected Paper by L.A. Zedeh, ed. R.R. Yager et al. John Wiley and Sons, 1987.

[4] L.A. Zadeh, "Fuzzy Logic," *IEEE Computer*, pp. 83-93, April, 1988.

[5] B. Kosko, "Fuzzy Entropy and Conditioning," *Information Sciences*, Vol. 40, pp. 165-174. 1986.

[6] D.Nguyen and B.Widrow, "The Thuck Backer-upper: an example of self-learning in neural networks," *Proceedings of IJCNN 1989*, vol. 2, pp. 357-363, June 1989.

[7] Seong-Gon Kong and Bart Kosko, "Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems," *Proceedings of IJCNN 1990,* Vol. 3. pp 349-358, June 1990.

Cerebellar Model Articulation Controller Neural Network — A Simple Fuzzy Expert System in Disguise?

P.A. Ramamoorthy, Song Huang
Department of Electrical & Computer Engineering,
University of Cincinnati, M.L. #30
Cincinnati, Ohio 45221-0030

ABSTRACT

Neural networks and fuzzy expert systems have attracted the attention of many researchers recently. In general, fuzzy logic uses verbal information for handling higher-order logical relations between inputs and outputs which are not crisply defined. On the other hand, neural ne'works are used to obtain information about systems from large input/output observations and training or learning procedures. Thus, fuzzy logic and neural networks solve the same problem using different formulations. In this paper, we provide some insights along these lines by comparing a particular neural network architecture, that is, the Cerebellar Model Articulation Controller (CMAC) with fuzzy expert systems. We show that CMAC is in reality a simplified version of a fuzzy expert system though the former has been derived independently from biology-related concepts. We further show that these equivalences can be exploited to arrive at superior system modeling techniques that will retain the important aspects of these two areas. Such an approach will have both the design by trainability aspect of CMAC architecture and the decision making based on fuzzy or imprecise data property of fuzzy expert systems.

INTRODUCTION

Research in artificial intelligence, neural networks and fuzzy expert systems has progressed in parallel lines with one or the other (at one time or another) considered as the best and catch-all-solution for all the problems in the world. In reality, each method has its own advantages, and drawbacks and can be applied to only certain types of problems. The strengths of neural networks are the ability to learn specific input-output mappings from large input/output data samples possibly corrupted by noise, greater degree of robustness and the ability to adapt or continue learning. The strengths of fuzzy expert systems are the ability to deal with fuzzy information and incomplete or imprecise data in a structured or logical way. On the negative side, in the case of neural networks, questions such as how one can decide on the number of layers or nodes in the network, how one can train large networks or incorporate some structural information that one may have into such a network and if the network is successful in solving a problem, how one can explain their performance etc. remain open. In the case of fuzzy expert systems, the questions are how one can obtain a fuzzy rule base or fuzzy associate memory (FAM) if no expert is available or if the expert is unable to put his/her knowledge into a form which can be processed by computers or what to do if such rules themselves are not perfect or optimal. Since both of these techniques implement the same task (that of functional mapping and we can regard "inferencing" as one specific category under this class), a fusion of the two concepts that retains the individual strengths while overcoming their individual drawbacks will have excellent applications in the real world.

In this paper, we use one particular neural network - Cerebellar Model Articulation Controller (CMAC) - as an example to show that the concepts of inferencing by large data samples through a well connected network (the case of neural networks), and the inferencing through a set of fuzzy rules as well as their fuzzy sets and membership functions (the case of fuzzy expert systems) are very similar. The similarity exists not only in the aspect of causality (conducting inferencing through structural ways), but also in the aspect of methodology (breaking input domains into smaller subsets, sparse and regular interconnections between input and output domain). In the particular case of the CMAC network, we also show that it is a simple version of fuzzy expert systems.

CMAC NEURAL NETWORKS

A neural network can be considered as a system that maps an input u, a vector of size N, into an output y, a vector of size M, by the function $f : u \rightarrow y$ [1]. The mapping is performed in the network or system by weighting each and every input, summing the results, subtracting a bias value and passing the result through a nonlinear function which may produce a binary or bipolar or continuous value (between -1 to 1) for each output. Thus, it can be noticed that a neural network is nothing but a non-linear network. The mapping function f is assumed to be unknown and is estimated from several numerical I/O samples (u_i, y_i) through the training procedure.

Above, we described a one-layer feed-forward model of a neural network. It is widely assumed that the Kolmogrov's theorem on functional approximation is a proof that a two-layer neural network is sufficient for approximating arbitrary non-linear systems given sufficient number of hidden nodes [2]. But, it is only an existence proof and does not tell us how to arrive at the network. In fact, there surfaced questions as to whether this theorem itself is applicable to the problem at hand [2], but we are not concerned about that issue here. From our perspective, a neural network is a non-linear system with interconnected neurons, which maps an input into the output via the non-linear function f, and the function f is not given or known but estimated from a set of numerical I/O samples.

The Cerebellar Model Articulation Controller (CMAC) neural network was introduced by Albus [3, 4, 5] and seems to be getting renewed attention through the work of Miller, et al., [6, 7], Ersu, et al., [8] and Moody [9]. CMAC has been suggested as an alternative for backpropagation networks [1] to achieve better performance [7]. Since backpropagation is basically a gradient decent technique, applied to a multilayer nonlinear network it needs a large computation time, converges slowly for large systems, and has an error surface which may contain local minima. The CMAC network contains a single linear feedforward network that has to be trained and hence does not require error propagation etc. and therefore can learn the mapping rather quickly.

If we go through the description of the CMAC network, we will find definitions such as input sensors, receptive fields, input generalizations, input quantization, threshold logic gates, state-space detectors, collisions, virtual addresses, random hashing functions and multiple field detectors. Figure 1 is a simple example of a CMAC neural network from Miller et al. [7]. It has two inputs and one output. The operation of a CMAC neural network can be explained in simple terms as follows: 1. Each input is represented in a non-weighted representation with one "on-off" or binary line for each quantization level (for example, if the input ranges from 0 to 255 and spans only integer values, we will have 256 lines). 2. A certain number of adjoining lines are combined into a single entity, called input sensors (input generalization). They may overlap, leading to the term offset (input quantization). 3. The input-generalizations from various inputs are connected to a set of AND gates, called state-space detectors, in a regular and sparse fashion in order to reduce the interconnections. 4. These state-space detectors are connected to a smaller set of OR gates, called multiple field detectors. The connections are determined by assigning virtual addresses to the AND gates and passing the addresses of the active AND gates through a random hashing function. 5. The outputs of these multiple field detectors are weighted to produce the final results. The weights are determined, or equivalently the mapping function f is determined, based on observed data pairs (u_i, y_i) and supervised learning. In Figure 1, the input sensor's receptive fields are of rectangular shape, i.e. the output of the input sensors are binary values. Miller, et al., recently modified the original CMAC architecture [10] where it is suggested that: 1. The input sensors implement local receptive fields with tapered sensitivity functions (that is the sensor output is 1.0 if the input is in the center of the receptive field, and the output decreases linearly towards 0.0

[1]Backpropagation refers to an approach used to train multilayer networks and can be applied to any network. Hence it is not correct to call the multilayer perceptron network as a backpropagation network. We use it here as it has become a common practice.

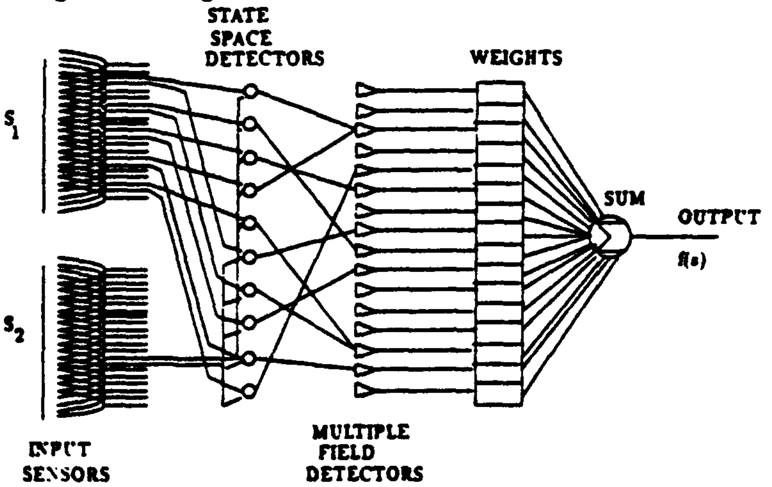
for inputs near the edges of the fields). 2. The state-space detectors can be considered as analog units (multiplication rather than logic AND gates) with the property that the unit output is 1.0 if all inputs are 1.0, while the unit output decreases to 0.0 if any input decreases to 0.0. 3. The multiple field detectors can be considered as simple summing units (rather than logic OR gates). The network output is then the sum of products of a certain number of non-zero multiple field detector outputs and corresponding weights. It is indicated that the modified CMAC architecture has better properties than the original CMAC because the modified version provides continuous instead of piece-wise function approximations..

FUZZY EXPERT SYSTEMS

Fuzzy logic developed by L.A. Zadeh [11, 12] can also be considered as a system for mapping (linear and nonlinear). The mappings are obtained by splitting the inputs and outputs into smaller and overlapping intervals or fuzzy sets, assigning membership functions (that indicates the degree of belonging of a particular input or output value to the various fuzzy sets of those inputs or outputs), fuzzy rule base or fuzzy associative memories (FAMs) that define the outputs fuzzy sets to which (expected final) outputs belong for inputs belonging to certain input fuzzy sets, and defuzzification procedure that provides the crisp output values from the selected output fuzzy sets and the membership functions.

To better explain the concept of how a fuzzy expert system conducts inferencing, an example from Kong and Kosko's paper is described [13]. The problem is to design a fuzzy controller to successfully back up a truck to a loading dock from any reasonable initial location. The inputs to the controller are the x, y positions of the truck and the truck orientation angle ϕ measured with respect to the x axis and the output is the steering signal θ. For a given command signal θ, the truck will move by a fixed distance and wait for the next signal from the controller. It is indicated that if enough space is given between the truck and the loading dock, then the y-position can be omitted as an input to the controller. The ranges of the inputs, x and ϕ, and the output, θ, are given as: ϕ: [0, 360]; x: [0, 100]; θ: [-30, 30].

Having identified the variables and their ranges, fuzzy subsets of the variables and their membership functions must be specified. Fuzzy subsets are simply linguistic terms and their numerical values corresponding to the input and output parameters. They are used to split the total range of the variables into smaller and overlapping ranges. Membership functions describe the degree of belonging of a particular input value to the various fuzzy sets and can have different shapes depending on the designer's preference or knowledge or experience. Triangular and trapezoidal shapes are used here for simpler calculation and better description of the problem. The membership functions of the variables for the truck-backer-upper problem are given in Figure 2.



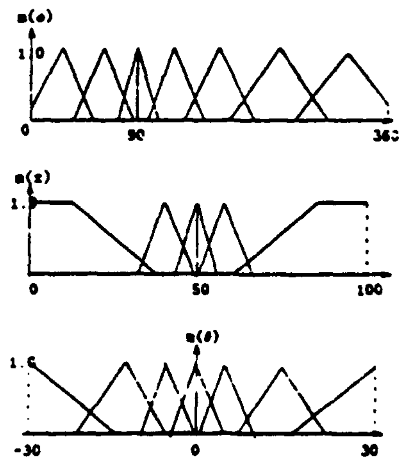


FIG. 1. A simple CMAC with 2 inputs and 1 output. FIG. 2. The membership function of TBU.

A fuzzy rule-base or fuzzy associative memories (FAMs) is a collection of fuzzy rules which define or describe the relationship between input fuzzy sets and output fuzzy sets. The rules are in the conventional IF–THEN form, with an antecedent part to describe the conditions and a consequent part to state the conclusions or actions. Figure 3 is the rule–

base for the truck-backer-upper problem defined by Kong and Kosko. This rule-base simply indicates the possible ranges for the output values given the ranges for the input values. The final crisp output values of a fuzzy expert system is determined using a procedure known as defuzzification process. The centroid method is mostly used as the defuzzification procedure and for the truck-backer-upper control problem as well. The trajectory of the truck starting from a given position backing up to the loading zone controlled by a fuzzy expert system is given in Figure 4.

		x-position				
		LE	LC	CE	RC	RI
angle (ϕ)	RB	PB	PB	PM	NB	NB
	RU	ZE	PS	PM	PB	PB
	RV	NB	NM	PS	PM	PB
	VE	NB	NM	ZE	PM	PB
	LV	NB	NM	NS	PM	PB
	LU	NB	NB	NM	NS	ZE
	LB	PB	PB	NM	NB	NB

FIG. 3. The rulebase of the TBU.

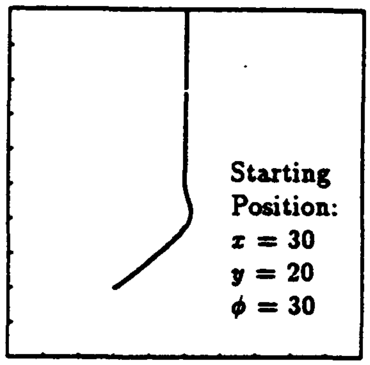


FIG. 4. A trajectory of the truck using a fuzzy controller.

MULTI-LAYER NN AND CMAC VS. FUZZY EXPERT SYSTEMS

From the definitions or steps involved in the implementation of CMAC neural networks (we discuss in terms of CMAC networks since it has been demonstrated that CMAC provides better performance than multi-layer networks) and fuzzy expert systems, it is clear that there is a good amount of similarities between the two approaches. The original CMAC can be considered to be a fuzzy expert system implementation where the receptive fields correspond to the fuzzy sets of the inputs with rectangular membership functions. The modified CMAC by Miller, et al., [7] modifies that membership function to a triangular one. In both, the input sensors correspond to joining many adjacent quantization levels of the input and thus correspond to the range selection for a fuzzy set. The output of the input sensors can then be thought of as pointers (binary valued) to those range of input values where a specific pointer becomes one only if the input falls in that range and zero otherwise. We can call such pointers as IFSPs (Input Fuzzy Set Pointers). Next, the outputs of these sensors (from different inputs) are ANDed (state space detectors) and the outputs of such units are ORed (multiple field detectors) in the original CMAC. It can be noted that the relationships between the outputs of these multiple field detectors (which are binary valued) and the outputs of the input sensors are expressible in boolean-sum-of-product form. If we define OFSPs (Output Fuzzy Set Pointers, binary valued) corresponding to fuzzy sets of outputs, the fuzzy rule base corresponds to a set of boolean-sum-of-product expressions for OFSPs in terms of IFSPs. Thus, the outputs of multiple field detectors in CMAC correspond to OFSP values in fuzzy expert systems. Finally, the output of a CMAC is simply a (linear) weighted version of these pointers[2], where as a fuzzy expert system uses more complex procedures (MIN,centroid, etc., operations) to produce the final output. Thus, the complexity is in the front end for a CMAC, where as it is uniformly distributed in the fuzzy expert system.

From the above descriptions, it can be inferred that a CMAC is indeed a simple fuzzy expert system in disguise. In the case of fuzzy expert systems, as we are able to solve the problem with very few fuzzy sets (of each variable). In the case of CMACs, as we do not make use of any available knowledge, we have to make use of a large number of fuzzy sets or input sensors. There are certain advantages in a CMAC approach over a fuzzy expert system. The use of a large number of multiple field detectors (or OFSPs) might allow the approximation

[2]It can be argued that in the modified CMAC, a triangular membership function is attached to each input sensor ranges and the exact value of the membership function at the given input values modifies the value of these pointers to something in the range 0 to 1.

of complex and arbitrary mappings[3]. Also, the training and linear stage for training makes the CMAC an attractive architecture for machine learning.

We can arrive at a new architecture (we call it "Neuro-Fuzzy" architecture) that will combine the properties of both neural networks (structure and trainability) and fuzzy systems (incorporation of apriori information, such as fuzzy sets and membership functions) without any of their drawbacks. Such a structure would be similar to the architecture shown in Figure 5. One would start with given I/O samples, make some assumptions about the number of fuzzy sets, their ranges and membership function shapes and train the various networks in Figure 5 in a systematic manner. That is, the number and the ranges for the fuzzy sets will be adapted based on I/O samples and once they are learned, the third network will be trained. The third network can again be decomposed into a number of smaller networks as shown in NN#2 of Figure 5. Since the data sets for each network will be comparatively smaller, the training can be achieved at a faster rate. Further, this architecture can be used as an implementational vehicle for fuzzy expert systems with a large number of input variables. The classical approach of implementation of fuzzy expert systems becomes too time consuming in such a case as the number of rules increases exponentially with an increase in the number of input variables. We are now looking into the various issues in the trainability of this architecture and other related issues. We give results of an example based on this technique in the next section.

NUMERICAL EXAMPLE

We tested the trainability of our neuro-fuzzy architecture using the truck-backer-upper controller problem. The truck-backer-upper fuzzy expert system controller has 7 input fuzzy sets for x-position, 5 sets for orientation angle ϕ, and 7 output fuzzy sets for the steering signal θ. Since more accurate results are required when the truck is in the center area or near the center area, we selected more samples for x-position around 50, and less samples otherwise. The training samples of ϕ are chosen in the same fashion. It led to 34 x-positions and 72 ϕ angles. Thus 2448 samples are used to train the controller. The y-positions are not used in training, thus simplifying the training process. There are 7 sub-networks in the system. The whole set of training samples is divided into 7 smaller groups according to their belongings to the output fuzzy sets. The largest group contained 826 training samples and the smallest one has 271 samples. Some samples are used in more than one subnet due to the overlapping of the fuzzy sets. This brought the total training samples for all subnets to 3624. We selected 10 neurons for the second layer of every subnet. The backpropagation algorithm was used for the training. The number of iterations for training varies from a few hundred (for smaller sample groups) to a few thousand (for larger groups). The average squared errors are from 0.0005 (for the centered or near center sets) to 0.0015 (for the extreme sets). The training samples were normalized to the range of -0.5 to 0.5. To show the robustness of this architecture, we tried two different approaches for the training. In one training procedure, we used the inputs, the desired outputs and input fuzzy set pointers as training samples. In the second method, we used membership functions of the input fuzzy sets rather their IFSPs as training samples. The truck trajectories produced by the trained networks are shown in Figures 6 and 7 respectively. As can be seen from the figures, the results are very encouraging.

CONCLUSION

In this paper, we compared the fuzzy expert systems with certain neural networks and show that a fuzzy expert system can be thought of as an advanced version of multilayer feedforward networks and CMAC networks. We show that a fuzzy expert system can be considered as a number of multilayer feedforward networks interconnected in a structured manner and the interconnection is defined by the concepts of fuzzy sets, fuzzy rules and

[3]Fuzzy expert system proponents might in turn argue that they can be achieved using few OFSPs and complex membership functions. Further, they may also question the need for such arbitrary mapping in real-world applications.

so on. More importantly, we show how these two powerful areas, that of neural networks and fuzzy expert systems, can be combined together to arrive at superior architectures and methodology to design adaptive intelligent systems.

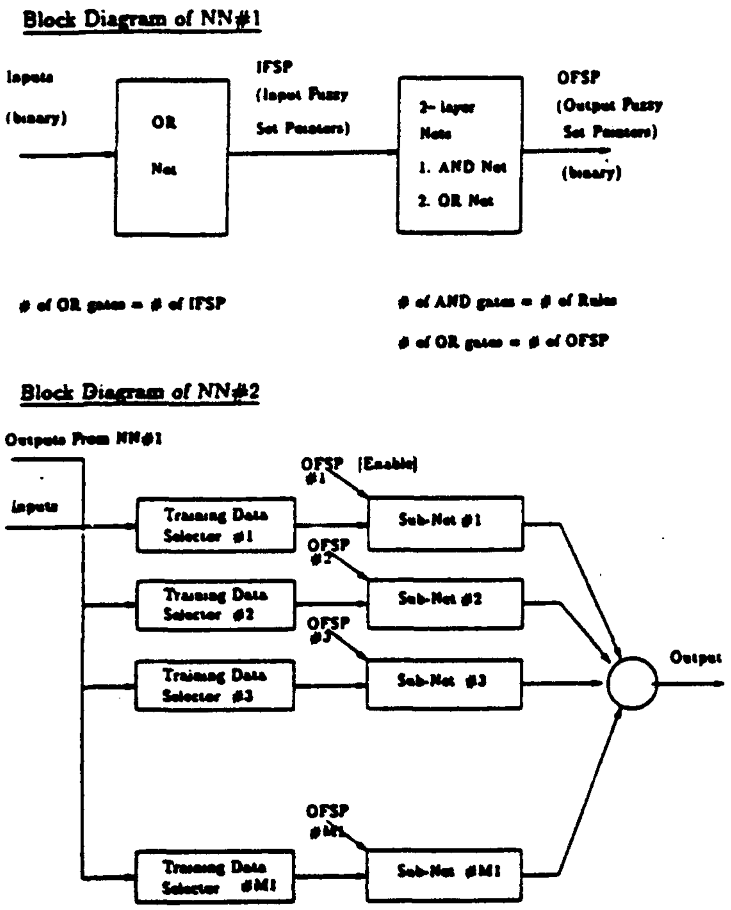


FIG. 5. A block diagram of the new Neuro-Fuzzy model.

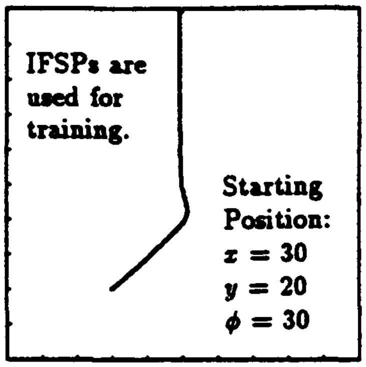


FIG. 6. A truck Trajectory using the Neuro-Fuzzy controller.

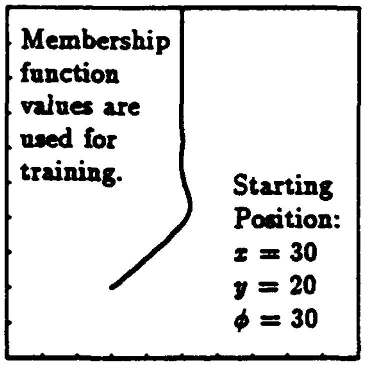


FIG. 7. A truck trajectory using the Neuro-Fuzzy controller.

REFERENCE

1. R. P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, 4-22 (April 1987).
2. G. Cybenko, "Approximation by superpositions of a sigmoidal function", Technical Report, University of Illinois (1988).
3. J.S. Albus, "A theory of cerebellar functions", Mathematical Biosciences, 10, 25-61 (1971).
4. J.S. Albus, "Theoretical and Experimental Aspects of a Cerebellar Model", PhD thesis, Univ. of Maryland (1972).
5. J.S. Albus, "Data storage in the cerebellar model articulation controller", J. of Dynamic Systems, Mearurement and Control, 228-233 (Sept. 1975).
6. W.T. Miller, "Non-linear learning controller for robotic manipulators", Proc. SPIE, Intelligent Robots and Computer Vision, 726, 416-423 (Oct. 1986).
7. W.T. Miller, F.H. Glanz, and L.G. Kraft, "CMAC: An associative neural network alternative to backpropagation", Proc. IEEE, 1561-1567 (Oct. 1990).
8. E.Ersu and H.Tolle, "Hierarchical learning control— an approach with neuron-like associative memories", Proc. IEEE Conf. on Neural Infor. Proc. Systems (Nov. 1988).
9. J.Mody, "Fast learning in multi-resolution hierarchies", in Advances in Neural Information Processing, edited by D.Touretzky (Morgan Kaufmann, 1989).
10. W.T. Miller, E.An, and F.H. Glanz, "The design of cmac neural networks for control", Proc. 6th Yale Workshop on Adaptive and Learning Systems (Aug. 1990).
11. L.A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes", IEEE Trans. Systems, Man and Cybernetics, SMC-3, 28-44 (1973).
12. L.A. Zadeh, "Fuzzy logic", IEEE Computer, 83-93 (April 1988).
13. S.Kong and B.Kosko, "Comparison of fuzzy and neural truck backer-upper control systems", Proc. of IJCNN, III, 349-358 (June 1990).

Proceedings of the IEEE International Conference on Systems Engineering
August, 1991, Dayton, Ohio

FUZZY EXPERT SYSTEMS VS. NEURAL NETWORKS — TRUCK BACKER-UPPER CONTROL REVISITED

P.A. Ramamoorthy and Song Huang

Department of Electrical & Computer Engineering,
University of Cincinnati, M.L. #30
Cincinnati, Ohio 45221-0030
FAX: 513-556-7326, Email: pramamoo@nest.ece.uc.edu

ABSTRACT

Research on neural networks and fuzzy logic have progressed on two independent paths. In general, fuzzy logic uses verbal information for handling higher-order logical relations between inputs and outputs which are not crisply defined. On the other hand, neural networks are used to obtain information about systems from large input/output observations and training or learning procedures. From these definitions, it appears that fuzzy logic and neural network fulfill two complementary functions. Hence, a merger of these two concepts could lead to powerful yet flexible knowledge processing tools. This paper provides some insights along these lines using the truck-backer-upper control problem. New network architectures by merging these two concepts and simulation results for the truck-back-upper problem using the new architecture are also shown in this paper.

INTRODUCTION

Both neural networks and fuzzy expert systems are systems that map an input u (a vector of size N) into an output y (a vector of size M) by the function $f : u \rightarrow y$. In a simple neural network, the mapping is performed in the system by weighing each and every inputs, summing the results, subtracting a bias value and passing the result through a non-linear function which may produce a binary or bipolar or continuous value [1,2]. Such networks may be cascaded to propagate the intermediate results to higher levels for more sophisticated problems. In the case of fuzzy expert systems, the ranges of the inputs and outputs are split into smaller and overlapping ranges or fuzzy sets. A fuzzy membership function is associated to each fuzzy subset. The mechanism governing the mapping from the input fuzzy sets to the output fuzzy sets is a collection of fuzzy rules — fuzzy rule base or fuzzy associative memories (FAM) [3,4]. The mapping from the inputs u to the outputs y is achieved through these fuzzy rules, the membership functions, and the defuzzification procedure.

There are similarities and differences between these two mapping systems. The similarities include provision for dealing with imprecise data or data corrupted by noise, having similar primitives or building blocks to produce nonlinear mapping (membership functions, fuzzy rules, MAX-MIN or centroid operations, vs. sigmoid functions in neural networks). The major difference is that the fuzzy expert systems use logic rules for inferencing while neural networks are data-driven. Therefore, fuzzy expert systems can be considered as *macroscopic tools for information processing*, whereas neural networks are microscopic in nature. The advantage of the neural networks is their ability to learn the mapping through training. The advantages of fuzzy expert systems are their ability to provide nonlinear mapping through the membership functions and fuzzy rules, and the ability to deal with fuzzy information and incomplete and/or imprecise data. By merging the advantages of these two systems, one can arrive at a more powerful yet flexible system for inferencing and learning. This concept will be explained through the use of results for the truck-backer-upper control problem.

PROBLEM DEFINITION

The truck backer-upper control is a typical nonlinear control problem where a controller to successfully back up a truck to a loading dock from any reasonable initial location has to be designed. Nguyen and Widrow [5] showed that a nonlinear controller using a two layer neural network architecture with 26 adaptive neural elements can be successfully trained. Recently, Kong and Kosko [6] compared the performance of such a neural network based controller with that of a controller based on a fuzzy expert system composed of 35 rules. They observed that even that simple fuzzy expert system lead to smoother trajectories than that produced by the two-layer neural network. If their observations are valid in general, it is desirable to arrive at a logical explanation for the differences in the performances. More importantly, as stated earlier, approaches that can retain the attractive properties of neural networks and at the same time obtain performances comparable to that of fuzzy expert systems need to be developed.

Figure 1 shows the loading zone of the truck-backer-upper problem and the input and the output variables of the system. If enough clearance is given between the truck and the loading dock, then the y-position can be omitted as an input to tune the controller. The ranges of the inputs, x-position and the truck orientation angle θ, and the output, steering signal ϕ, are given as:

$$\phi: [0, 360];\ x: [0, 100];\ \theta: [-30, 30]$$

Having identified the variables and their ranges, fuzzy subsets of the variables must be specified. Fuzzy subsets are simply linguistic terms and their numerical values corresponding to the input and output parameters. They are used to split the total range of the variables into smaller and overlapping ranges. The next step is the selection or identification of the membership functions associated with the various fuzzy subsets. The membership functions simply describe the degree of association of a particular input or output value to the fuzzy subsets belonging to that input or output parameter. The membership functions of the variables for the truck-backer-upper problem are given in Figure 2. Membership functions can have different shapes depending on the designer's preference or knowledge or experience. Triangular and trapezoidal shapes are used here for simpler calculation and better description of the problem.

A fuzzy rule-base or fuzzy associative memory (FAM) is a collection of fuzzy rules which define or describe the relationship between input fuzzy sets and output fuzzy sets. The rules are in the conventional IF-THEN form, with an antecedent part to describe the conditions and a consequent part to state the conclusions or actions. Figure 3 is the rule-base for the truck- backer-upper problem defined by Kong and Kosko. This rule-base simply indicates the possible ranges for the output values given the input values. The final crisp output values of a fuzzy expert system arare determined using a procedure known as defuzzification process. Centroid method is mostly used as the defuzzification procedure and for the truck-backer-upper control problem as well. The trajectory of the truck from a given position backing up to the loading zone controlled by a fuzzy expert system is given in Figure 4. The trajectory of a truck produced by a two layer neural network accordingly is reproduced as Figure 5 (Kong and Kosko). The controller has 24 hidden neurons and trained by over three thousand samples.

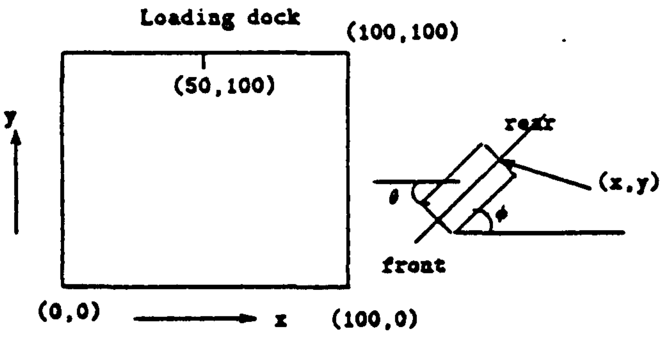


Figure 1. The loading zone of the TBU problem.

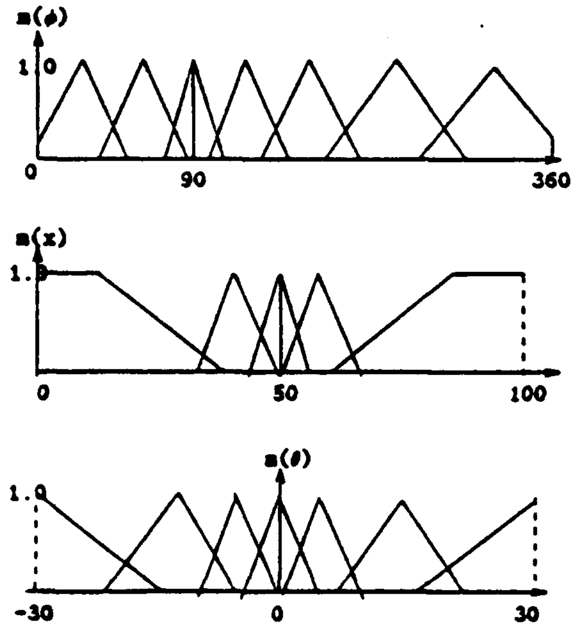


Figure 2. The membership functions of the TBU problem.

angle	x-position				
(φ)	LE	LC	CE	RC	RI
RB	PB	PB	PM	NB	NB
RU	ZE	PS	PM	PB	PB
RV	NB	NM	PS	PM	PB
VE	NB	NM	ZE	PM	PB
LV	NB	NM	NS	PM	PB
LU	NB	NB	NM	NS	ZE
LB	PB	PB	NM	NB	NB

Figure 3. The rulebase of the TBU problem.

The conclusion drawn by Kong and Kosko is that a fuzzy expert system controller is superior compared to that of a neural network. If both fuzzy expert systems and neural networks provide robust nonlinear mapping, the difference in the results of these two systems should not be significant. One may argue that the differences may be due to the size of the network, input/output samples used for training and the number of iterations. (But such practical constraints are bound to exist). On the other hand, one can argue that even if such parameters (as the number of nodes) are increased to the maximum practical limit, the neural network performance may not be comparable simply because the fuzzy expert systems use more information in a structured way. We take the later attitude and proceed from these to arrive at a methodology that will combine the best of both worlds, trainability in neural networks and better mapping property of fuzzy expert systems. This will become clear in the next section.

NETWORK REPRESENTATION OF FUZZY EXPERT SYSTEM

Let us examine the steps in implementing a fuzzy expert system. Given the exact input values, we first determine the fuzzy sets to which these inputs belong. If we use an unweighted binary representation for the inputs (i.e. Q lines carrying 0 or 1 for Q quantization levels) and one bit per fuzzy set to indicate if a particular input has fallen under that fuzzy set (we call this as input fuzzy set pointers), this step can be represented as an OR network as shown in Figure 6a. The inferencing from the rule-base can be thought of as turning "on or off" of the bits denoting output fuzzy sets based on which input fuzzy set has been selected. Thus, the inferencing can be represented by a two-layer AND-OR (or sum of product) network (the second block in Figure 6a). The defuzzification process can be represented as yet another network whose inputs are the output fuzzy set pointers and the membership function values for a given input value. The outputs of this network will be the final outputs of the fuzzy expert system. This network can be subdivided into smaller networks, each of which can be a 2 or 3 layer perceptron networks as in Figure 6b. The number of sub-networks is equal to the number of output fuzzy sets in all the output variables and each sub-network generates an output if and only if that network is enabled by the corresponding output fuzzy set pointer. The outputs from the selected networks are averaged to find the final outputs.

From the above discussion, it is obvious that these three blocks constitute a network representation of a fuzzy expert system. Thus, it is conceivable that the performance will be poor (degraded) if we try to represent the tasks of these three individual blocks in a single 2 or 3 layer neural network. One may argue that a 3 layer network is sufficient to represent any nonlinear mapping. But this is only true from a theoretical point of view but may not hold from an engineering point of view.

Having identified the network implementation of a fuzzy expert system, we can use this structure and any additional knowledge (besides large input/output samples) that we may have to find the actual interconnections/weights of the individual blocks. For example, if we assume that the fuzzy sets (of inputs and outputs), the corresponding membership functions and the fuzzy rules are known but the defuzzification procedure is unknown, the architecture of the first two blocks can be arrived at very easily and the third block can be trained using the output fuzzy set pointers, membership function values and the desired outputs. Such a training can be very fast as we have represented the third block as a number of smaller sub-blocks. Many variations of this procedure are possible depending upon what kind of information is available. We will be discussing all such possibilities in another paper.

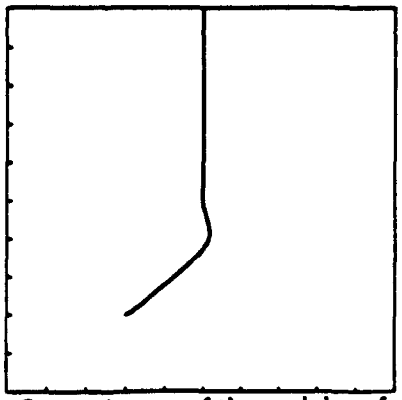


Figure 4. One trajectory of the truck by a fuzzy controller.

Let us now show the results based on the above approach. There is no training for the first two blocks of Figure 6a as indicated before. The third block is trained by 1) using both input and output fuzzy set pointers, inputs x and ϕ values and output values and 2) using output fuzzy pointers, input membership function values and input, output values. The data needed for the training were generated using the original fuzzy controller.

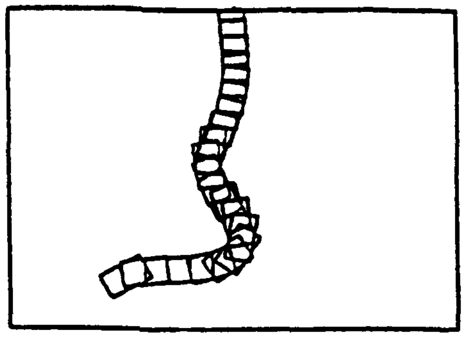


Figure 5. One trajectory of the truck by a NN controller.

The truck-backer-upper fuzzy expert system controller has 7 input fuzzy sets for x-position, 5 sets for orientation angle ϕ, and 7 output fuzzy sets for the steering signal θ. Since more accurate results are demanded when the truck is in the center area or near center area, we selected more samples for x-position around 50, and less samples to the

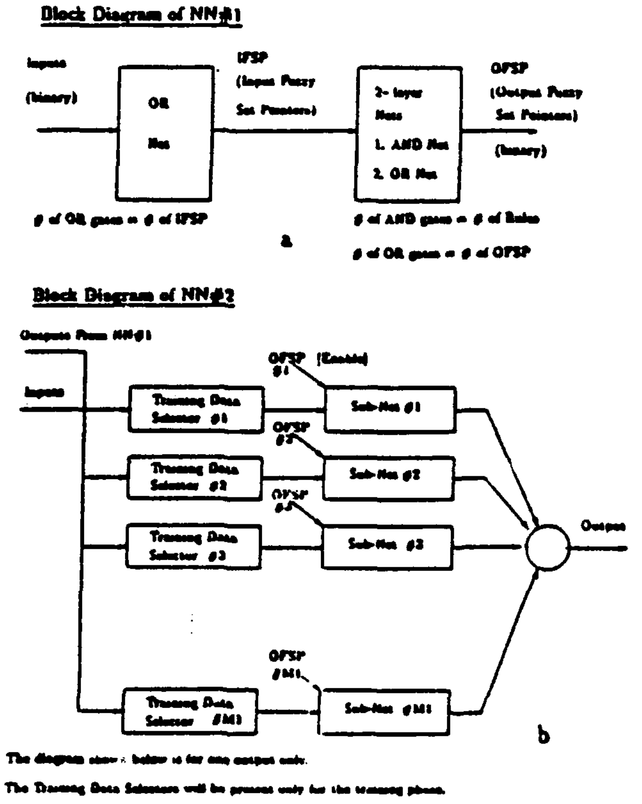


Figure 6. Block diagrams of the FES-NN system.

extremes. The training samples of ϕ are chosen in the same fashion. It led to 34 x-positions and 72 ϕ angles. Thus 2448 samples are used to train the controller. The y-positions are not used in training, thus simplifying the training process. There are 7 sub-networks in the system. The whole set of training samples are divided into 7 smaller groups according to their belongings to the output fuzzy sets. The largest group contained 826 training samples and the smallest one has 271 samples. Some samples are used in more than one subnets due to the overlapping of the fuzzy sets. this brought the total training samples for all the subnets to 3624. There are 10 second-layer neurons for every subnet. The backpropagation algorithm was used for the training. The number of iterations for training varied from few hundred (for smaller sample groups) to few thousand (for larger groups). The average square errors are from 0.0005 (for the

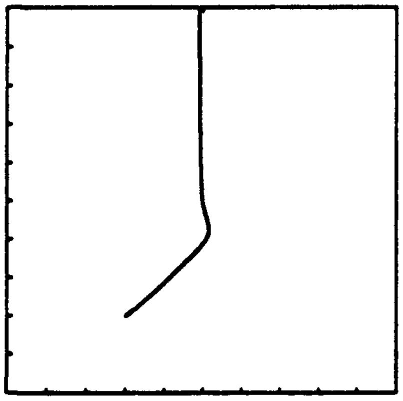


Figure 7. One trajectory by the fuzzy-NN controller.

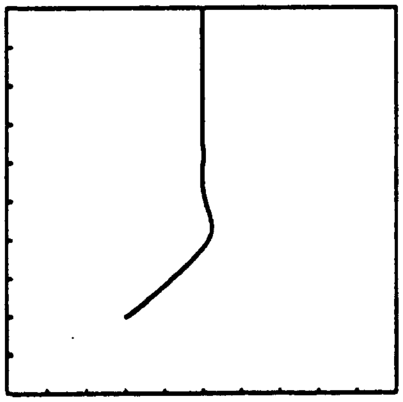


Figure 8. One trajectory by the fuzzy-NN controller.

centered or near center sets) to 0.0015 (for the extreme sets). The training samples were normalized to the range of -0.5 to 0.5. As stated before, either membership function values or input fuzzy set pointers are used with the input variables for training the nets. To show the robustness of this architecture, we did the training by using these two different sets of information. A truck trajectory produced by using the neural network corresponding to case 1) is shown in Figure 7, and Figure 8 shows one trajectory corresponding to case 2). It can be noted that both methods produces superior results than the one generated by a two layer neural network as shown by Kong and Kosko for this particular initial condition. Further, our own efforts to train a two layer network with 20 hidden nodes with the same input/output data was only marginally successful and the performance of that controller was very poor.

CONCLUSIONS

A fuzzy expert system is characterized by six attributes: 1) fuzzy input variables, 2) fuzzy output variables, 3) fuzzy sets of the input and output variables, 4) membership functions corresponding to the fuzzy sets, 5) fuzzy rules connecting the input fuzzy sets and the output fuzzy sets, and 6) methodology for defuzzification of the fuzzy output. In all real world problems, we will have (or we can infer) useful information about the first four attributes based on the problem at hand. Of these four attributes, only two (number 1 and 2) are presently used in the design of neural network architectures. We showed that by using the other two attributes, one can arrive at new neural network architectures that will provide superior performance and also smaller networks that can be trained rather easily. This concept is proved through the use of the design of a new controller for the truck-backer-upper control problem.

In this work, we considered the training of only one of the three blocks of our new network. However, if information about fuzzy subsets and the rules are not available, one can train the other two blocks as well using some initial knowledge about the input and the output variables. Once these blocks are trained, we can sort of pull out the fuzzy rules for further examination and modification. Work along these lines is being carried out presently.

ACKNOWLEDGEMENT

This research work has been supported by a grant NAG #3960 from NASA Lewis Research Center, by UC-NASA Center Grant and a grant N 00014-89-J-1633 from Office of Naval Research.

REFERENCES

1. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing — Explorations in the Microstructure of Cognition*, Cambridge, MA. MIT Press, 1986.

2. R.P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

3. L.A. Zadeh, "Fuzzy Logic," *IEEE Computer*, pp.83-93, April 1988.

4. B. Kosko, "Fuzzy associative memories," in *Fuzzy Expert Systems* (A. Kandel, ed.), New York: Addison-Wesley, 1987.

5. D. Nguyen and B. Widrow, "The truck backer-upper: an example of self-learning in neural networks," *Preceedings of IJCNN 1989*, vol.2, pp. 357-363, June 1989.

6. S. Kong and B.Kosko, "Comparision of fuzzy and neural truck backer-upper control systems." *Proceedings of IJCNN 1990*, vol.3, pp. 349-358, June 1990.